

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ЕЁ ПРИМЕНЕНИЕ

Новое в жизни, науке, технике

Подписная научно-популярная серия

Издается ежемесячно с 1988 г.

Язык программирования Си

1991

4

Новое
в жизни,
науке,
технике

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА
И ЕЁ ПРИМЕНЕНИЕ

Подписная
научно-
популярная
серия

4/1991

Издается
ежемесячно
с 1988 г.

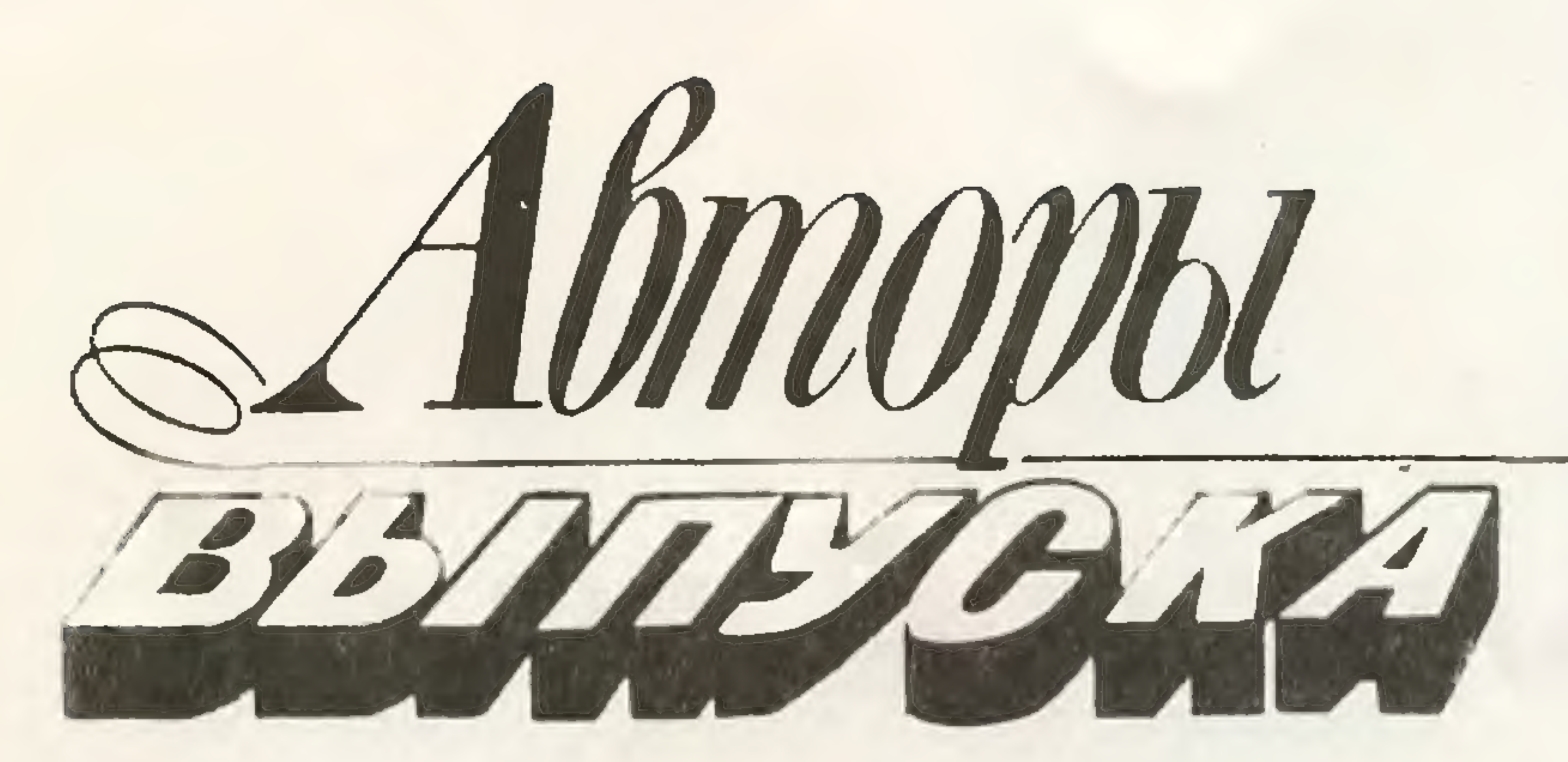
ЯЗЫК ПРОГРАММИРОВАНИЯ СИ

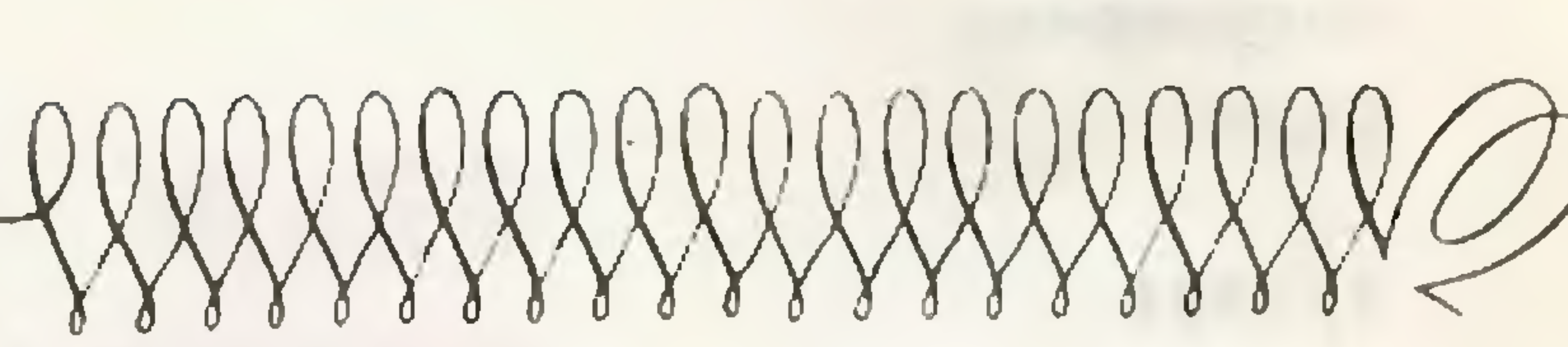
В номере:



Москва
Издательство
"Знание"
1991

ББК 32.85
Я 53





ДУКАРЕВИЧ Григорий Борисович — аспирант Московского института электронного машиностроения, программист.

КОВАЛЕВ Андрей Александрович — инженер-программист. Профессиональные интересы — экспертные системы, базы данных и их применение в информационно-справочных системах и САПР.

РЕДАКТОР Б. М. ВАСИЛЬЕВ

ISBN 5-07-001720-9



Начиная изучать язык Си, рассмотрим его основные достоинства и недостатки. Это поможет нам лучше узнать положительные стороны языка и подготовит к трудностям, подстерегающим нас при изучении и использовании его на практике.

Г.Б.Дукаревич

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ СИ

ПОЧЕМУ ЯЗЫК НАЗЫВАЕТСЯ СИ?

Си (по-английски С) - это язык программирования, широко распространенный в настоящее время среди системных программистов и разработчиков прикладных программ. История появления и развития языка Си связана с операционной системой UNIX. В отличие от других языков, язык Си появился как инструментальный язык в процессе работы над операционной системой UNIX Кена Томпсона из научно-исследовательской лаборатории фирмы Bell Telephone Laboratories. Она была расположена в г. Марри-Хилл штата Нью-Джерси. В 1969 году Томпсон начал работать над этой системой для мини-ЭВМ PDP-7 фирмы DEC. В 1972 году Денисом Ритчи и Кеном Томпсоном был создан язык программирования Си. В нем сочетались лучшие свойства языка Ассемблер и языков высокого уровня. От Ассемблера были взяты гибкие и эффективные средства работы с памятью, от языков высокого уровня - широкий набор управляющих конструкций, возможность работы со сложными структурами данных, гибкие средства ввода/вывода. Сегодня компилятор с языка Си есть практически на любом компьютере в любой операционной системе. 95% операционой системы UNIX и почти все прикладное обеспечение для нее написано на языке Си.

Почему язык называетя Си? Если вы заметили, это название соответствует третьей букве английского алфавита. Совпадение не случайно и имеет свою историю. Каждый автор, создавая язык программирования, должен придумать ему имя. Так, один из языков, появившихся в семидесятых годах назывался APL (A Programing Language) - язык программирования А (Эй). Так была занята первая буква алфавита. Во время работы над операционной системой UNIX для PDP-7 Томпсоном был создан язык программирования В (Би), который оказал сильное влияние на следующий язык, разработанный Ритчи и Томпсоном для тех же целей. Показывая эту преемственность, авторы назвали язык С (Си). Как знать, может быть, следующую букву использует кто-нибудь из тех, кто читает сейчас этот выпуск.

Достоинства языка Си

Эффективность. По компактности и скорости выполнения программы на языке Си приближаются к программам, написанным на языке Ассемблер.

Мощность. Язык Си содержит большой набор современных управляющих конструкций и способов агрегатирования данных, который может удовлетворить самых взыскательных пользователей.

Способность поддерживать технологию структурного программирования. Управляющие конструкции языка Си согласуются с технологией структурного программирования.

Способность поддерживать разработку модульных программ. Основной программной единицей является функция, есть возможность создавать многофайловые программы.

Мобильность. Программы, написанные на языке Си, могут быть легко перенесены на другой компьютер в другую операционную систему.

Лаконичность. Программы на языке Си получаются короче, чем на других языках программирования. При этом не теряется, а увеличивается наглядность и ясность программ.

Недостатки языка Си

Язык Си очень удобен в использовании и, на наш взгляд, обладает лишь одним существенным недостатком: он предъявляет достаточно высокие требования к квалификации использующего его программиста. В связи с этим нам хочется дать несколько существенных рекомендаций.

1. Не начинайте знакомство с программированием с языка Си. Возьмите лучше какой-нибудь более простой и легкий в изучении язык. Больше всего, по нашему мнению, для первого знакомства с программированием подходит Паскаль (Pascal). Это может быть и какой-нибудь другой язык, например Модула (Modula) или Бейсик (Basic).

2. При изучении Си желательно иметь представление о структуре и работе компьютера.

3. Большую помощь и более глубокое понимание идей Си, как языка системного программирования, обеспечат хотя бы минимальное знание языка Ассемблер.

4. В процессе изучения необходимо очень осторожно пользоваться не до конца изученными возможностями, правилами умолчания и т. д. и не полагаться на свою программистскую интуицию (она может подвести).

Если вас не испугают и не остановят эти замечания, то, изучив язык Си, вы приобретете надежный, сильный инструмент. Он поможет вам справится с любой, даже самой сложной задачей. Там, где другие языки могут окзаться недостаточно подходящими или совсем неприменимыми, язык Си поможет вам достичь цели кратчайшим путем. Чтобы все это осуществилось, вам необходимо затратить немало усилий.

К сожалению, ограниченный обьем не позволит нам пройти весь путь от первого знакомства с языком до совершенного владения этим мощным инструментом, но мы попытаемся вместе сделать первые шаги и указать вам дорогу, по которой вы уже сами будете шагать навстречу знанию.

Пример для знакомства

Знакомство с языком Си начнем с примера простейшей программы на этом языке.

Программа 1.

```
/* Моя первая программа */          /* 1 */
#include <stdio.h>                  /* 2 */
                                    /* 3 */
main()                              /* 4 */
{                                   /* 5 */
    printf("Hello, Friend!\n");     /* 6 */
        /* Здравствуй, Друг! */
}                                   /* 7 */
```

Результат работы нашей первой программы будет выглядеть так:
Hello, Friend!

Давайте рассмотрим эту программу подробнее. Первая строка содержит комментарий, т. е. текст, который игнорируется компилятором и предназначается только для программиста и его коллег. Комментарии используют для того, чтобы сделать программу более понятной. Большое количество полезных комментариев говорит о хорошем стиле программирования и высокой квалификации программиста. Комментарием в программе на Си будет любая последовательность символов, заключенная между знаками/* */. Заметим, что между символами / и *, а также * и / пробелы недопустимы,

иначе компилятор не поймет, что это начало или конец комментария. Комментарии могут располагаться как на одной строке, так и на нескольких, например:

```
/************************
* Это тоже комментарий *
************************/
```

Появиться комментарии могут везде, где только может возникнуть пробел в программе. При этом они приравниваются к пробельным символам. Не могут они появиться внутри идентификаторов, чисел, многосимвольных знаков операций или других служебных комбинаций символов. Недопустимы также вложенные комментарии, т. е. комментарии внутри комментариев, например:

```
/* Это /* вложенный */ комментарий */
```

В данном случае первая комбинация */ будет воспринята как конец комментария, и на оставшуюся часть комментария будет выдана ошибка.

Вторая строка содержит так называемую директиву препроцессора. В отличие от других языков программирования обработка программы на Си осуществляется в два этапа: сначала программа обрабатывается препроцессором, т. е. некоторой программой, осуществляющей выполнение специальных директив, а только затем она компилируется, т. е. обрабатывается программой, которая переводит текст программы с языка, понятного человеку, на язык машины. Как правило, оба эти этапа объединяются в одну программу и ее называют просто компилятором.

Строки, содержащие директивы препроцессора, начинающиеся с символа номера (#). Директива #include осуществляет подстановку вместо себя текста, указанного в директиве файла (в данном случае файла с именем stdio.h). Заключение имени файла в угловые скобки (<stdio.h>) говорит о том, что поиск этого файла будет осуществляться в системном каталоге Си. В языке Си файлы с расширением .h называются файлами-заголовками (Header File). Они содержат описания переменных, функций, типов и т. д., которые используются многими программами. В данном случае в файле stdio.h содержатся описания, необходимые для использования стандартной библиотеки ввода/вывода языка Си. Имя файла получилось от сокращения Standard Input/Output (стандартный ввод/вывод). Этот файл мы будем включать практически перед всеми нашими программами.

Кроме включения файлов, препроцессор осуществляет еще подстановку значений, условную компиляцию и некоторые другие полезные действия.

Третья строка оставлена нами пустой для улучшения восприятия текста.

Четвертая строка содержит заголовок определения функции, в котором указано имя функции (в нашем примере main). В отличие от других языков программирования в Си есть только один вид подпрограмм - это функции. Даже основная программа, как ее называют в некоторых языках, тоже является функцией. Имя основной программы, т. е. той, с которой начнется выполнение в языке Си, должно быть обязательно main. Поскольку у нас только одна функция, то другого имени мы ей дать не можем. Скобки после имени указывают на то, что это функция; в них описываются параметры. Поскольку в нашей программе параметров нет, то скобки пустые, но они обязательно должны присутствовать, опускать их нельзя.

Открывающая фигурная скобка ({) в пятой строке говорит о том, что это начало тела функции.

Тело функции состоит только из одного оператора. Он выводит на экран сообщение «Hello, Friend!» (6 строка) и переводит курсор в начало следующей строки (по символу «/n»). Он выполняет это действие, обращаясь к стандартной функции вывода printf (печать форматная) из стандартой библиотеки ввода/вывода. Для того чтобы использовать эту функцию, мы включили в нашу программу файл заголовка stdio.h. Обращение к функции осуществляется так же, как и в других языках

программирования; указывается имя вызываемой функции, вслед за именем в круглых скобках указываются параметры. В нашей программе у функции printf один параметр - строковая константа «Hello, Friend!/n». Поскольку нас не интересует значение, возвращаемое функцией, то мы его никак не используем, и оно теряется. Заканчивается оператор точкой с запятой (;). Этот сивол в языке Си в отличие от других языков программирования не разделяет операторы, а является частью оператора, поэтому точка с запятой здесь обязательна.

Закрывающая фигурная скобка (}) в седьмой строке указывает на конец тела функции, а в нашем случае и на конец программы вообще.

Мы рассмотрели подробно строка за-строкой всю программу. Взгляните на нее еще раз, все ли вам понятно? Если это не так, то лучше вернитесь и разберите ее еще раз, чтобы у вас не осталось никаких неясностей. Было бы очень хорошо , если бы вы могли попробовать на компьютере все программы, которые мы будем разбирать. Только практика позволит вам приобрести прочные знания и уверенность в себе.

Для того чтобы программу на языке Си можно было выполнить на вашем компьютере, ее сначала нужно обработать двумя программами:

1. Компилятором, который переведет написанный вами текст в машинные команды (только их может понимать компьютер), а заодно и проверит правильность написания программы.

2. Сборщиком, который объединит вашу программу с другими функциями, например ввода/вывода и т. п., и представит ее в форме, в которой она уже может быть запущена на выполнение с помощью команд операционной системы.

ЗАБЕГАЯ ВПЕРЕД

Рассматривая программу 1 в предыдущем разделе, мы уже изучили некоторые операторы языка Си. Теперь познакомимся с новыми директивами препроцессора и стандартными функцями ввода/вывода, которые мы будем использовать в примерах.

Директива определения символической константы

#define <идентификатор> <значение>

определяет значение идентификатора. Далее во всей программе идентификатор в тексте будет заменяться на указанное значение. Этот механизм используется очень часто для определения констант, например:

#define MAXLEN 100

В тексте идентификатор MAXLEN в процессе препроцессирования будет заменён на значение 100. Такой механизм позволяет легко модифицировать программу.

Ввод и вывод символа

В языке Си предусмотрены стандартные функции ввода/вывода. Самым простым механизмом ввода является чтение по одному символу из так называемого стандартного ввода, который связан обычно с клавиатурой. Специальными средствами он может быть переназначен для ввода с другого устройства или из файла. Переназначение стандартного ввода - это простое и в тоже время достаточно мощное средство и очень широко используемое в программах на языке Си.

Получение символа осуществляется с помощью функции getchar() (get character - получить символ), которая возвращает очередной входной символ. Эта функция без параметров, но, как мы уже говорили, пустые скобки опускать нельзя. Например,

ch = getchar();

присваивает переменной ch очередной символ, полученный со стандартного ввода. В случае возникновения ситуации «конец файла» функция getchar() возвращает значение EOF, которое, как правило, определено в файле stdio.h и не совпадает ни с одним символом.

Аналогичная программа используется для вывода одного символа на стандартный вывод, который, как правило, связан с экраном. Он так же как и стандартный

ввод, специальными средствами может быть переназначен на другое устройство, например принтер или файл. Эта функция имеет один параметр - выводимый символ. Записывается она следующим образом:

putchar(ch)

(put character - выдать символ), а ch - символ, который будет выдан на стандартный вывод. Например:

putchar('A');

выдаст на стандартный вывод заглавную букву А.

Форматированный ввод/вывод

Программы форматированного ввода и вывода также содержатся в стандартной библиотеке и доступны после включения файла stdio.h. Эти программы осуществляют необходимое преобразование числовых величин при их выводе и вводе.

Программа printf (print format - форматная печать), которая уже встречалась нам в программе 1, осуществляет печать данных на стандартный вывод в соответствии с указанным форматом представления данных. Обращение к программе осуществляется следующим образом:

printf(<формат>, <аргумент1>, <аргумент2>,...)

Формат представляет собой строковую константу, содержащую объекты двух типов: обычные символы, которые просто копируются в выходной поток, и спецификации преобразования, каждая из которых вызывает преобразование и печать очередного аргумента. Аргументами могут быть любые выражения, значения которых мы хотим вывести.

Здесь мы рассмотрим только основные спецификации преобразования:

%d - вывод значения целого типа,
%g - вывод значения вещественного типа,
%c - вывод одного символа,
%s - вывод строки символов.

Например:

```
printf("Целое=%d, Вещественное=%g, Символ=%c, Строка=%s/n",
        100+23, 3.14159, 'A', "String");
```

Результатом выполнения этого оператора будет:

Целое=123, Вещественное=3.14159, Символ=A, Строка=String

Аналогично программа scanf (scan format - форматный ввод) осуществляет ввод данных из стандартного ввода и преобразование их во внутреннее представление в программе. Вызов программы записывается следующим образом:

scanf(<формат>, <аргумент1>, <аргумент2>,...);

Формат может содержать символы трех типов:

пробелы, символы табуляции и перехода на новую строку - эти символы игнорируются;

обычные символы: считается, что они должны совпадать с очередными не пустыми символами во входном потоке;

спецификации преобразования, указывающие необходимые преобразования вводимых данных для очередного аргумента.

Спецификации преобразования функции scanf полностью совпадают со спецификациями преобразования функции printf. Аргументами программы ввода могут быть только простые переменные или имена символьных массивов для ввода символьных строк, причем перед именами простых переменных обязательно должен стоять знак амперсенд (&). Например,

```
scanf("Int=%d, Real=%g, Char=%c, String=%s/n", &i,
        &r, &c, s);
```

где

i - переменная целого типа,

r - переменная вещественного типа,
c - переменная символьного типа,
s - массив символьного типа.

Пусть входная строка выглядит следующим образом:

Int=123, Real=3.14159, Char=A, String=String

тогда переменная i получит значение 123, переменная r - значение 3.14159, переменная c - значение 'A' и в массив s будет занесена строка «String».

ПЕРЕВОД ДЮЙМОВ В САНТИМЕТРЫ

Рассмотрим программу, которая печатает таблицу соответствия дюймов сантиметрам. Соотношение между дюймами и сантиметрами следующее: 1 дюйм = 2,54 сантиметра.

Для определенности будем считать, что в таблице должны быть значения, начиная с одного до десяти дюймов с шагом один дюйм.

Сначала опишем нашу задачу на русском языке. Для ее решения необходимо выполнить следующую последовательность действий:

Вывести заголовок таблицы.
Установить текущее значение, равное начальному значению.
Пока текущее значение не больше конечного значения,
повторять:
    вычислить значение в сантиметрах, соответствующее
    текущему значению;
    вывести текущее значение и соответствующее ему
    значение в сантиметрах;
    увеличить текущее значение на шаг приращения.

Записанный нами текст является не чем иным, как алгоритмом, в соответствии с которым должна выполняться наша программа. Для того чтобы получить программу, нам нужно перевести этот алгоритм на язык программирования. Давайте посмотрим на алгоритм и выясним, можем ли мы перевести каждое предложение в операторы языка Си. Первое предложение начинается со слова «Вывести ...», значит, ему будет соответствовать вызов функции вывода языка Си (например, printf(...)). Предложение, начинающееся со слов «Установить ... равное ...» соответствует оператору присваивания. После слова «установить» указана переменная, которая получает новое значение, а после слова «равное» указан способ его вычисления. Предложение «Пока ... повторять ...» соответствует конструкции цикла, в которой тело цикла (указанное после слова «повторять») повторяется до тех пор, пока остается истинным условие (указанное после слова «пока»). Предложение, начинающееся со слова «вычислить ...», может соответствовать любому вычислению или вызову подпрограммы. В нашем случае необходимо вычисление по формуле, поэтому фраза «вычислить...» будет соответствовать оператору присваивания. Предложение, начинающееся со слова «увеличить ...», соответствует операции увеличения значения языка Си (или операции присваивания).

Мы рассмотрели управляющие конструкции, встретившиеся в нашем алгоритме. Теперь давайте разберем, какие переменные использовались нами в алгоритме, и определим назначение каждой из них. Сначала просто перечислим их:

текущее значение;
начальное значение;
конечное значение;
значение в сантиметрах;
шаг приращения.

В нашем алгоритме мы использовали пять переменных, которые по своему назначению можно разделить на две группы. В первую войдут переменные, соответствующие исходным данным, необходимым для получения таблицы: начальному значению, конечному значению и шагу приращения. Значения этих переменных должны быть известны до начала выполнения алгоритма. Во вторую группу войдут рабочие переменные, которые используются для организации вычислений. У нас две такие переменные - текущее значение и значение в сантиметрах. Они вычисляются на каждой итерации цикла и служат для формирования таблицы. Все переменные, используемые в нашем алгоритме, имеют вещественные значения, т. е. применяются для хранения дробных чисел.

Поскольку в языке Си имена переменных так же, как и операторы, должны быть записаны английскими буквами, то нам нужно придумать эти имена, которые можно записать по-английски. Для этого поступим следующим образом. Переведем названия переменных на английский язык, а затем возьмем сокращение этих фраз, с одной стороны, не очень длинное, а с другой - достаточно мнемоничное. Необходимо помнить о том, что для именования переменных, а также и других элементов языка, например, функции, типы, структуры, метки и т. д., используются идентификаторы. В языке Си идентификатором называется последовательность букв, цифр и символов подчеркивания, начинающаяся с буквы или символа подчеркивания. В идентификаторе допускается любое количество символов, но только первые 31 символ используются компилятором для распознавания идентификатора. Не рекомендуется начинать имена с символа подчеркивания (_), потому что с него начинаются имена внутренних идентификаторов Си. Идентификаторы не могутсовпадать по написанию с ключевыми словами, список которых приведен в конце статьи.

Имя в алгоритме	Перевод	Имя переменной
текущее значение	current value	CurVal
начальное значение	start value	StartVal
конечное значение	end value	EndVal
значение в сантиметрах	value in centimetre	ValInCen
шаг приращения	step	Step

Теперь мы можем записать программу на Си по приведенному алгоритму.

Программа 2

```
/* Перевод дюймов в сантиметры */

#include <stdio.h>

main()
{
    float StartVal                          /* начальное значение       */
    float EndVal;                           /* конечное значение        */
    float Step;                             /* шаг приращения           */

    float CurVal;                           /* текущее значение         */
    float ValInCen;                         /* значение в сантиметрах */

    StartVal = 1.;
    EndVal  = 10.;
```

```
    Step      = 1.;

    /* вывод заголовка таблицы */
    printf("Дюймы Сантиметры/n");

    CurVal = StartVal;
    while  (CurVal <= EndVal) {
          ValInCen = CurVal * 2.54;
          printf("%5.0f %7.3f/n", CurVal, ValInCen);
          CurVal += Step;
    }

}
```

Теперь давайте рассмотрим программу и разберем незнакомые нам операторы. Эта программа, как и предыдущая, начинается с комментария, в котором коротко говорится о назначении программы. Затем идет директива препроцессора #include <stdio.h>, которая включает в текст программы описание функций стандартной библиотеки ввода/вывода. Наша программа состоит из одной функции с именем main() без параметров (мы уже говорили о том, что так обязательно должна называться функция Си, с которой начинается выполнение программы).

Дальше идут описания переменных, используемых в программе. Все переменные имеют один и тот же тип float - вещественный тип данных. Его значения занимают четыре байта памяти, лежат в диапазоне от 1E-37 до 1E+37 и представляют значения с точностью 1E-5. _Кроме указанного типа, в языке Си используется еще один встроенный тип данных double - вещественный тип данных двойной точности, значения которого занимают четыре байта памяти, лежат в диапазоне от 1E-307 до 1E+307 и представляют значения с точностью 1E-9. Вещественные константы в языке Си принадлежат типу double.

Описание переменных выглядит следующим образом: сначала пишется спецификатор типа, а затем через запятую (,) список переменных. Заканчивается описание, как и все операторы языка Си, точкой с запятой (;).

<спецификатор типа> <идентификатор>, <идентификатор> ...;

Тип данных переменных задается спецификатором типа (мы знаем пока лишь две из них - float и double). По мере рассмотрения новых примеров мы разберем и другие спецификаторы типов. Все переменные, используемые в программе на Си, обязательно должны быть описаны. Все описания должны предшествовать выполняемым операторам.

После описаний переменных идут выполняемые операторы. В нашей программе выполняемые операторы начинаются с трех операторов присваивания, в которых переменные получают значения соответствующих вещественных констант.

Отличительная особенность языка Си та, что любое выражение, заканчивающееся точкой с запятой (;), является оператором и что присваивание в нем реализовано не как оператор, а как операция. Это значит, что операция присваивания может встречаться в выражении несколько раз, в том числе и внутри других операций.

В качестве знака операции присваивания используется знак равенства (=). Операция присваивания имеет два операнда. Левым операндом может являться переменная, а правым - любое выражение языка Си. Очевидно, что соблюдение порядка операндов обязательно, и их перестановка приведет к ошибке.

Выражение, образуемое операцией присваивания, записывается следующим образом:

<переменная> = <выражение>

Значением этого выражения является значение левого операнда после выполнения операции присваивания. Кроме того, она имеет побочный эффект, который заключается в том, что изменяется значение переменной, указанной в левой части выражения.

Если в выражении несколько операций присваивания идут подряд, то они выполняются справа налево (обратите на это внимание).

Примеры:

```
x = (y * 23) / h;
count = count + 1;
z = (m = 5) + 11;
a = b = c = 0;
a = (b = (c = 0));
```

Если использовать оператор-выражение, содержащее только операцию присваивания, то в этом случае ее запись и смысл полностью будут соответствовать оператору присваивания в других языках программирования. Первое время вы можете пользоваться только такой записью.

Следующая строка программы 2 - оператор цикла с предусловием. Называется он так потому, что условие выполнения цикла проверяется перед выполнением тела цикла. В общем виде оператор записывается следующим образом:

```
while (<выражение>){
        <оператор1>
          ...
        <оператор N>
}
```

Операторы 1-N называются телом цикла. Выполнение оператора происходит следующим образом. Сначала вычисляется значение выражения. Если оно истинно, то выполняется тело цикла (операторы 1-N). Затем опять вычисляется значение выражения и т. д. до тех пор, пока значение выражения истинно. Как только оно станет ложным, выполнение программы продолжится с оператора, следующего за циклом. Если выражение ложно с самого начала, то тело цикла не выполняется ни разу.

На самом деле в описании языка Си приводится описание управляющих конструкций, отличное от предлагаемого нами, т.е. телом управляющей конструкции указывается только один оператор, например:

```
while (<выражение>)
        <оператор>
```

Если нам необходимо, чтобы тело управляющей конструкции содержало несколько операторов, то вводится понятие составного оператора, состоящего из одного или более операторов любого типа, заключенных в фигурные скобки ({}). После закрывающей фигурной скобки точка с запятой не ставится.

Составной оператор может использоваться везде, где может использоваться простой оператор.

Например:

```
{
        x = 1;
        y = 2;
        z = 3;
}
```

Вариант управляющих конструкций с составным оператором является более общим и используется чаще. Его применение ведет к уменьшению количества ошибок, вносимых при модификации программы. По этой причине в дальнейшем мы будем приводить описание управляющих конструкций с составным оператором, не забывая при этом, что во всех управляющих конструкци-

Таблица 1

Знак	Название операции
==	равно
!=	не равно
<	меньше
<=	меньше или равно
>	больше
>=	больше или равно

Таблица 2

Знак	Название операции
!	логическое «НЕ»
&&	логическое «И»
||	логическое «ИЛИ»

ях вместо составного оператора может стоять простой, и тогда фигурные скобки не пишутся.

Выражением в операторе цикла может быть любое выражение, допустимое в Си. Однако в этом языке нет специальных средств для работы с логическими значениями, поэтому при определении значения выражения поступают следующим образом. Если значение выражения равно нулю, то считают, что оно имеет логическое значение ложь. Если же оно отлично от нуля, то истина.

В выражениях, употребляемых в заголовке цикла, часто используются операции отношения и логические операции, поэтому здесь мы приводим запись и определение их в языке Си.

Операции отношения приведены в таблице 1. Логические операции показаны в таблице 2.

Выполнение логических операций «И» и «ИЛИ» отличается от выполнения этих операций в других языках.

Выполнение операции «И» производится следующим образом. Вначале вычисляется значение первого операнда. Если оно ложно, то и все выражение получает значение ложь. Если же оно истинно, то вычисляется значение второго операнда, и значение всего выражения определяется по второму операнду.

Аналогично осуществляется и выполнение операции «ИЛИ». Вычисляется значение первого операнда. Если оно истинно, то и все выражение получает значение истина; если же оно ложно, то вычисляется значение второго операнда, и значение всего выражения определяется по нему.

При использовании оператора цикла с предусловием необходимо помнить о том, что в теле цикла должны быть операторы, влияющие на значение выражения, иначе цикл никогда не закончится.

Тело оператора цикла содержит в нашем случае три оператора. Первый, оператор присваивания, в котором вычисляется значение в сантиметрах, соответствующее текущему значению в дюймах. Для этого переменной, содержащей значение в сантиметрах, присваивается значение произведения текущего значения и 2,54.

Си располагает тем же набором арифметических операций, что и другие языки программирования. Этот набор арифметических операций приведен в таблице 3.

Таблица 3

Знак	Название операции
-	изменение знака
+	сложение
-	вычитание
*	умножение
/	деление
%	взятие по модулю (вычисление остатка от деления)

Все операции, кроме первой, бинарные, т.е. имеют два операнда. Операция изменения знака - унарная, т. е. имеет один операнд.

Вторым оператором тела цикла является вызов функции печати. При этом происходит вывод текущего значения в дюймах, соответствующего ему значения в сантиметрах и переход в начало следующей строки. Значения переменных выводятся как вещественные числа.

К третьему относится оператор увеличения значения переменной на значение выражения, указанного справа от знака равенства. Эта операция имеет следующий формат:

<переменная> += <выражение>

и эквивалентна следующему оператору присваивания:

<переменная> = <переменная> + (<выражение>)

Результаты работы программы выглядят следующим образом:

Дюймы	Сантиметры
1	2.540
2	5.080
3	7.620
4	10.160
5	12.700
6	15.240
7	17.780
8	20.320
9	22.860
10	25.400

НАХОЖДЕНИЕ МИНИМАЛЬНОГО И МАКСИМАЛЬНОГО ЭЛЕМЕНТОВ ПОСЛЕДОВАТЕЛЬНОСТИ

Рассмотрим задачу, в которой дана последовательность вещественных чисел. Требуется найти ее минимальный и максимальный элементы. Считаем, что сначала вводится количество элементов последовательности, а затем сами элементы.

Давайте теперь более формально запишем условия задачи.

ДАНО:

количество элементов последовательности

элемент1, элемент2, ..

ТРЕБУЕТСЯ:

минимальный элемент последовательности;

максимальный элемент последовательности.

СВЯЗЬ:

минимальный элемент последовательности - это элемент, который не больше любого элемента последовательности;

максимальный элемент последовательности - это элемент, который не меньше любого элемента последовательности.

Теперь проведем анализ задачи по формальному описанию, для того чтобы принять решения, которые помогут нам разработать алгоритм.

АНАЛИЗ:

1. Поиск минимального и максимального элементов аналогичен с точностью до знака сравнения, поэтому достаточно провести анализ только для половины задачи.

2. Для того чтобы определить, что элемент является минимальным (максимальным), его необходимо сравнить со всеми элементами последовательности.

3. Поскольку сравнивать минимальный (максимальный) элемент с другим достаточно только один раз, то нет необходимости хранить в памяти все эле-

менты последовательности, т. е. вполне можно использовать следующую схему обработки:

Для всех элементов последовательности повторять:
ввести очередной элемент;
сравнить очередной элемент с минимальным (максимальным).

4. На протяжении всего цикла обработки для получения правильного результата должно сохраняться условие, что минимальный (максимальный) элемент не больше (не меньше) всех уже просмотренных элементов. Рассмотрим действия, необходимые для сохранения этого условия, в двух возможных вариантах: очередной элемент больше или равен минимальному (меньше или равен максимальному) и меньше минимального (больше максимального). В первом случае очередной элемент можно просто пропустить, при этом условие сохранится. Во втором случае необходимо заменить минимальный (максимальный) элемент на текущий. Так как минимальный (максимальный) элемент не больше (не меньше) всех предшествующих, а текущий меньше минимального, следовательно, текущий не больше всех предшествующих. Значит, при замене минимального (максимального) элемента на текущий условие сохранится.

5. При вводе исходных данных необходимо помнить о том, что количество может быть только положительным числом и определение минимального и максимального элементов последовательности возможно только в том случае, если в последовательности не менее одного элемента.

6. Нам осталось решить только один вопрос, какие начальные значения будут иметь минимальный и максимальный элементы. Представляется наиболее разумным в качестве минимального и максимального элементов последовательности выбрать первый элемент.

Из приведенной формальной постановки задачи и анализа уже видны некоторые переменные, которые будут использоваться в алгоритме. Например, количество элементов и текущий элемент, которые являются входными данными, минимальный и максимальный элементы, которые являются выходными данными, т. е. результатом работы программы. Кроме того, при составлении алгоритма могут добавиться еще и внутренние переменные, необходимые для решения задачи.

Теперь можно переходить к разработке алгоритма.

РАЗРАБОТКА АЛГОРИТМА:

Ввести и проверить количество элементов последовательности.
Для всех элементов последовательности повторять:
ввести очередной элемент;
если это первый элемент, то
установить минимальный и максимальный
элементы, равными первому элементу последовательности;
определить новый минимальный элемент;
определить новый максимальный элемент.
Вывести минимальный и максимальный элементы.

Приведенный вариант алгоритма описывает все действия, которые необходимо выполнить для решения задачи, он раскрывает некоторые из них недостаточно подробно для того, чтобы был очевиден их перевод на язык программирования. С другой стороны, если все очень подробно раскрыть в этом варианте, то он получится очень сложным и более трудным для понимания. Можно предложить следующий выход из этой ситуации. Записывать алгоритм небольшим количеством действий, а те из них, которые не очевидны, детализировать затем отдельно. Такой подход в программировании называют или нисходящей разработкой, или методом пошаговой детализации, или методом пошаговых уточнений. В этом случае на очередном этапе раскрывается только один пункт плана предыдущего уровня. Именно поэтому данный этап мы назвали не алгоритмом, а разработкой алгоритма. Он показывает, что процесс уточнения алгоритма возможен на нескольких уровнях. Этим же методом мы воспользуемся дальше при разра-

ботке более сложных программ. Чтобы показать, какой из предыдущих пунктов и как мы уточняем, напишем его вначале в фигурных скобках ({}). Кроме того, вы, наверное, уже заметили, что некоторые слова в наших описаниях алгоритмов соответствуют ключевым словам управляющих конструкций языков программирования. Например, такие, как: пока - повторять, если - то - иначе, для - повторять и др. Для более легкого перевода алгоритма на язык Си выделим эти управляющие конструкции отступами и укажем явно конец каждой управляющей конструкции словами: конец повторения, конец, если и т. д. Это позволит нам четко указать тело соответствующей конструкции.

В приведенном фрагменте алгоритма мы использовали управляющую конструкцию для - повторять. Давайте рассмотрим ее подробнее.

Для всех элементов последовательности повторять:
<действие>
конец повторения.

Она говорит о том, что <действие> должно быть выполнено столько раз, сколько элементов в последовательности. Их количество нам известно, следовательно, цикл повторится заданное число раз. При переводе на Си можно пользоваться эквивалентом конструкции для - повторять, использующим уже известную нам конструкцию цикла с предусловием:

Установить счетчик равным единице.
Пока счетчик не больше количества элементов повторять:
<действие>
увеличить счетчик;
конец повторения.

При такой реализации цикла у нас добавляется еще одна рабочая переменная целого типа - «счетчик».

Кроме рассмотренной реализации, может быть и другая, более подходящая для данной ситуации, основанная на конструкции цикла со счетчиком.

Для счетчика от единицы до количества элементов с шагом
единица повторять:
<действие>
конец повторения.

Если шаг увеличения переменной цикла (счетчика) равен единице, то об этом можно не писать.

Конструкции цикла со счетчиком соответствует оператор цикла со счетчиком языка Си, который в общем виде записывается так:

```
for (<выражение1>; <выражение2>; <выражение3>) {
    <оператор1>
    ...
    <операторN>
}
```

<выражение1> описывает инициализацию цикла;

<выражение2> - условие продолжения выполнения цикла;

<выражение3> - действие, выполняемое после выполнения тела цикла, перед очередной проверкой условия продолжения (выражения2).

Выполнение оператора происходит следующим образом. Сначала вычисляется значение <выражения1>. Затем вычисляется значение <выражения2>, если оно истинно, то выполняется тело цикла, и после него вычисляется значение <выражения3>, после этого опять вычисляется значение <выражения2>, если оно истинно, то выполняется очередная итерация цикла и так далее до тех пор, пока значение <выражения2> не станет ложным, тогда управление передается на оператор, следующий за оператором цикла. Заметим, что <выражение3> вычисляется после каждого выполнения тела цикла.

Оператор цикла со счетчиком эквивалентен следующей последовательности операторов:

```
<выражение1>;
while (<выражение2>) {
    <тело цикла>
    <выражение3>
}
```

Любое из трех выражений или два или все три могут отсутствовать, однако разделяющие их точки с запятой опускать нельзя. Если опущено <выражение2>, то оно считается всегда истинным.

Оператор for (;;) представляет собой бесконечный цикл, эквивалентный while (1).

Пример:

```
for (x = 1; x < 7; x++) {
    printf("%d/n", power(x, 2));
}
```

Тот же самый оператор можно записать одним из следующих способов:

Способ 1

```
x = 1;
for (; x < 7; x++) {
    printf("%d/n", power(x, 2));
}
```

Способ 2

```
for (x = 1; x < 7;) {
    printf("%d/n", power(x, 2));
    x++;
}
```

Способ 3

```
x = 1;
for (; x < 7;) {
    printf("%d/n", power(x, 2));
    x++;
}
```

Итак, начинаем детализировать неочевидные шаги приведенного алгоритма.

```
{Ввести и проверить количество элементов последовательности}
Повторять:
        вывести запрос;
        ввести количество элементов последовательности;
при допустимом значении количества элементов выход;
        вывести сообщение о недопустимом количестве элементов;
конец повторения.
```

В этом фрагменте мы использовали управляющую конструкцию цикла с выходом

```
Повторять:
    <действие1>
при <условие> выход;
    <действие2>
конец повторения.
```

Эта конструкция выполняется следующим образом: сначала <действие1>, затем проверяется <условие>. Если оно истинно, то происходит выход из цикла, т. е. его выполнение завершается. Дальше выполняются действия, следующие за словами «конец повторения». Если же условие ложно, то выполняется <дей-

ствие2>, потом <действие1>, а затем снова проверяется <условие> и т. д. до тех пор, пока <условие> не станет истинным.

В языке Си нет конструкции цикла, соответствующей рассмотренной нами, но ее легко можно построить самим, используя оператор прерывания выполнения, который применяется для прекращения выполнения ближайшего внешнего оператора цикла (while, do, for) или оператора выбора (switch). Записывается оператор следующим образом:

```
break;
```

При выполнении этого оператора управление передается оператору, следующему за заканчиваемым. Одно из назначений этого оператора - закончить выполнение цикла при возникновении некоторого условия в теле цикла. В этом случае он используется совместно с условным оператором. При этом конструкция цикла с выходом на языке Си может быть записана так:

```
while (1) {
    <действие1>
if (<условие>) break;
    <действие2>
}
```

Единица в заголовке цикла while задает всегда истинное условие, т. е. фактически мы используем бесконечный цикл с предусловием и в его теле помещаем оператор прерывания цикла при выполнении <условия>.

В языке Си наряду с оператором прекращения конструкции (break) используется двойственный ему оператор продолжения.

Оператор продолжения передает управление в начало ближайшего внешнего оператора циклов while, do, for, вызывая начало следующей итерации. Оператор записывается в следующем виде:

```
continue;
```

По действию он противоположен оператору break.

Например, следующий фрагмент вводит n целых чисел и распечатывает только положительные:

```
int i, val;
for (i =0; i < n; i++) {
    scanf("%d", val);
if (val <= 0) continue;
    printf("%d/t", val);
}
```

Рассмотренная конструкция цикла с выходом очень часто используется для ввода значений, которые должны удовлетворять заданным условиям.

Считаем, что дальше уточнять этот фрагмент нецелесообразно, в нем уже каждая фраза может быть переведена на соответствующий оператор языка Си без затруднений, поэтому переходим к уточнению следующего шага:

```
{определить новый минимальный элемент}
Если текущий элемент меньше минимального элемента, то
    установить минимальный элемент равным текущему;
конец если.
```

И аналогично для определения максимума.

```
{определить новый максимальный элемент}
Если текущий элемент больше максимального,
    то установить максимальный элемент равным текущему;
конец если.
```

Таким образом, мы детализировали весь алгоритм до состояния, когда каждую фразу можно переписать на язык программирования. Теперь для получения конечного варианта алгоритма соберем все фрагменты вместе, а

чтобы текст остался легко понимаемым, сохраним текст, заключенный в фигурные скобки, в качестве комментариев перед блоками алгоритма.

```
АЛГОРИТМ:
  {Определить минимальный и максимальный элементы}
  {последовательности }
  {Ввести и проверить количество элементов последовательности}
  Повторять:
    вывести запрос;
    ввести  количество элементов последовательности;
  при допустимом значении количества элементов выход;
    вывести сообщение о недопустимом количестве элементов;
  конец повторения.
  Для счетчика от единицы до количества элементов повторять:
    ввести очередной элемент;
    если счетчик равен единице, то
      установить максимальный и минимальный элементы
      равными первому элементу последовательности;
    конец если.
    {определить новый минимальный элемент}
    Если текущий элемент меньше минимального элемента, то
      установить минимальный элемент равным текущему
    конец если.
    {определить новый максимальный элемент}
    Если текущий элемент больше максимального, то
      установить максимальный элемент равным текущему;
    конец если.
  конец повторения.
  Вывести минимальный и максимальный элементы.
```

Нам известны все используемые переменные, теперь придумаем для них имена.

Имя в алгоритме	Перевод	Имя переменной
Количество элементов	Number of elemets	Number
Текущий элемент	Current element	Current
Минимальный элемент	Minimum element	Minimum
Максимальный элемент	Maximum element	Maximum
Счетчик	Counter	Counter

Переменные Number и Counter - целого типа, остальные (Current, Minimum, Maximum) - вещественного типа.

Данные целого типа служат для хранения целых чисел различной длины, причем они могут быть как знаковые, так и беззнаковые. Параметры целых типов приведены в табл. 4.

Таблица 4

Параметры целых типов

Тип	Размер	Диапазон
(signed) char	1 байт	-128.. 127
unsigned char	1 байт	0.. 255
(signed) short (int)	2 байта	-32 768.. 32 767
unsigned short (int)	2 байта	0.. 65 535
(signed) long (int)	4 байта	-2 147 483 648..2 147 483 647
(signed) long (int)	4 байта	-2 147 483 648..2 147 483 647
unsigned long (int)	4 байта	0..4 294 967 295

В таблице нет типа данных int, потому что в зависимости от реализации он соответствует или типу short, или типу long. Как правило, на 16-разрядных компьютерах он соответствует short, а на 32-разрядных он соответствует типу long.

Ключевые слова signed и int указаны в скобках потому, что они подставляются по умолчанию и их можно опускать. Мы указали их для единообразия. В некоторых ранних реализациях ключевого слова signed может не быть.

Тип char мы включили в эту таблицу не случайно, так как данные символьного типа могут использоваться в арифметических выражениях, и тогда они будут трактоваться как числа согласно табл. 4.

Далее приведем текст соответствующей алгоритму программы.

```
ПРОГРАММА:
/* Определить минимальный и максимальный элементы */
/* последовательности                             */
#include <stdio.h>

main()
{
   int    Number;    /* количество элементов  */
   int    Counter;   /* счетчик               */
   float Current;    /* текущий элемент       */
   float Minimum;    /* минимальный элемент   */
   float Maximum;    /* максимальный элемент  */

   /* Ввести и проверить количество элементов последовательности */
   while (1) {
      printf("Введите количество элементов последовательности>");
      scanf("%d", &Number);
   if  (Number > 0) break;
      printf("Недопустимое количество элементов/n");

   }

   /* Определить минимальный и максимальный элементы */
   for(Counter = 1; Counter <= Number; Counter++) {
      printf("Введите %d-й элемент>", Counter);
      scanf("%g", &Current);
      if (Counter == 1) {
         Minimum = Maximum = Current;
      }
      /* определить новый минимальный элемент */
      if (Current < Minimum) {
         Minimum = Current;
      }
      /* определить новый максимальный элемент */
      if (Current > Maximum) {
         Maximum = Current;
      }
      Counter++;
   }
   /* Вывести минимальный и максимальный элементы */
   printf("Минимальный элемент: %g/n", Minimum);
   printf("Максимальный элемент: %g/n", Maximum);
}
```

Рассмотрим новые операторы языка Си, которые мы использовали в этой программе. Начнем с условного оператора.

Условный оператор в языке Си, как и во многих других языках, имеет два варианта: сокращенный и полный. Мы последовательно рассмотрим тот и другой.

Сокращенный условный оператор

```
if (<выражение>) {
    <оператор1>
    ...
    <операторN>
}
```

Если выражение истинно (т. е. значение выражения отлично от нуля), то выполняется <оператор1>..<операторN>. Если выражение ложно (т. е. значение выражения равно нулю), то ничего не делается.

Полный условный оператор

В отличие от сокращенного оператора полный содержит действия для обоих случаев (истинного и ложного значения выражения).

```
if (<выражение>) {
    <оператор1>
    ...
    <операторN>
} else {
    <операторN+1>
    ...
    <операторM>
}
```

Если выражение истинно, то выполняются <оператор1>...<операторN>, если нет, то <операторN+1>...<операторM>.

Например:

```
if (a > b) {
        c = a;
} else {
        c = b;
}
```

Так как else-часть условного оператора может отсутствовать, то во вложенных условных операторах и в этом случае, если телом оператора является не составной, а одиночный оператор, может возникнуть неоднозначность. Считается, что else-часть относится к ближайшему предыдущему if в том же блоке, не имеющему else-части.

Например:

```
if (x > 1)
if (y == 2) z = 5;
else z = 6;
```

В этом случае else-часть относится ко второму if. Для того чтобы это сразу стало видно, мы рекомендуем использовать фигурные скобки и отступы. Этот оператор тогда будет выглядеть так:

```
if (x > 1) {
    if (y == 2) {
        z = 5;
    } else {
        z = 6;
    }
}
```

Во многих языках программирования в последнее время используется еще один вариант условного оператора, в котором в зависимости от выполненного условия выбирается соответствующее действие, например:

```
if (x < -10) {
    y = f1(x
} else if (x < 0) {
    y = f2(x);
} else if (x == 0) {
    y = f3(x);
} else if (x < 10) {
    y = f4(x);
} else {
    y = f5(x);
}
```

В этом случае последовательно проверяются условия, и как только некоторое условие выполняется, выполняется и соответствующее действие. Все остальные проверки и действия пропускаются. В общем виде такую конструкцию можно записать так:

```
if (<выражение1>)
    <операторы1>
} else if (<выражение2>) {
    <операторы2>
  .
  .
  .
} else {
    <операторыN>
}
```

Если истинно <выражение1>, то выполняются <операторы1>, если истино <выражение2>, выполняются <операторы2> и т. д. В том случае, если ложны все выражения после if, то выполняются <операторыN>. Else-часть может и отсутствовать. В этом случае, если ложны все выражения, то ничего выполняться не будет.

Последний рассмотренный вариант условного оператора является более мощным аналогом оператора switch, рассматриваемого далее.

Кроме условного оператора, мы использовали еще два варианта оператора присваивания. Первый

```
Minimum = Maximum = Current;
```

говорит о том, что переменным Minimum и Maximum присваивается значение переменной Current. В общем случае это запишется так:

```
<переменная1> = <переменная2> =...= <переменнаяN> = <выражение>;
```

Все переменные получат одинаковое значение, равное значению <выражения>.

Второй условно относится к группе операций присваивания и образуется операцией инкремента.

Операции инкремента и декремента осуществляют соответственно увеличение или уменьшение значения переменной на единицу. В отличие от остальных операций присваивания эти операции унарные (т. е. у них только один операнд). Эти операции обозначаются соответственно двумя знаками плюс или минус, следующими один за другим (++ и —). В зависимости от положения знака операции различают пред- и постинкремент и декремент.

Выражение, образуемое операцией прединкремент, записывается следующим образом:

```
++<переменная>
```

Значением выражения является значение переменной ПОСЛЕ инкремента. Кроме того, операция имеет побочный эффект, заключающийся в увеличении значения переменной на единицу.

Примеры:
++count;
h = ++a / 2;

Во втором примере значение переменной *a* увеличивается на единицу и делится на 2. Если перед выполнением оператора значение переменной *a* было 5, то после его выполнения значение переменной h будет 3 (6/2), а значение переменной *a* - 6. Такой оператор соответствует двум следующим операторам присваивания:
a = a + 1;
h = a / 2;

Выражение, образуемое операцией постинкремент, записывается следующим образом:
<переменная>++
Значением выражения в этом случае будет значение переменной ДО инкремента. Побочный эффект тот же - увеличение значения переменной на единицу.
Примеры:
count++;
= x++ * 3;

Во втором примере сначала значение переменной *x* используется для вычисления значения переменной *m* и только потом увеличивается на единицу. Допустим, что перед выполнением этого оператора переменная *x* имела значение 6, тогда после его выполнения значение переменной *m* будет 18 (6*3), а значение переменной *x* будет 7. Приведенный оператор соответствует двум следующим операторам присваивания:

m = x * 3;
x = x + 1;

Операторам
++<переменная>;
и
<переменная>++;
соответствует следующий оператор присваивания:
<переменная> = <переменная> + 1;

Отсюда следует, что если операции пред- или постинкремента используются как отдельный оператор, то они имеют одинаковый смысл. Разница между этими операциями проявляется только при использовании их в других выражениях. Результаты работы программы могут выглядеть, например, так:

Введите количество элементов последовательности>10
Введите 1-й элемент>1.3
Введите 2-й элемент>4
Введите 3-й элемент>-3
Введите 4-й элемент>-0.5
Введите 5-й элемент>109.4
Введите 6-й элемент>11e
Введите 7-й элемент>123.45
Введите 8-й элемент>44.6
Введите 9-й элемент>9.0
Введите 10-й элемент>0.345
Минимальный элемент: -3
Максимальный элемент: 12345

ВЫЧИСЛЕНИЕ КВАДРАТНОГО КОРНЯ

ЗАДАЧА:

Разработать программу вычисления квадратного корня из заданного числа с заданной точностью методом Ньютона.

Метод Ньютона:

Если известно некоторое приближение квадратного корня заданного числа, то более точное приближение можно получить по следующей формуле:

(число/приближение + приближение)/2

ДАНО:

число

точность вычисления корня

ТРЕБУЕТСЯ:

значение корня

СВЯЗЬ:

следующее приближение = (число/приближение + приближение)/2

АНАЛИЗ:

1. В качестве начального приближения можно взять любое значение, но для определенности выберем 1.

2. При вводе исходных данных необходимо проверить, чтобы и число и точность были больше нуля.

3. Условием окончания вычисления станет проверка абсолютного значения отношения без единицы заданного числа к квадрату корня. Должно быть не больше заданной точности.

Поскольку алгоритм в этой задаче простой, то мы запишем его сразу, пропустив этап разработки. Ввод и проверку исходных данных будем осуществлять аналогично тому, как мы это делали в предыдущей задаче.

```
АЛГОРИТМ
Ввести и проверить исходные данные.
Установить значение корня , равное единице.
Повторять:
    Установить следующее приближение корня, равное значению
    выражения (число/корень + корень)/2;
пока абсолютное значение отношения без единицу заданного
числа к квадрату корня не больше заданной точности.
Вывести результат.
```

Составим таблицу имен переменных:

Имя в алгоритме	Перевод	Имя переменной
число	number	Number
точность	precision	Precision
корень	root	Root

Все переменные вещественные.

ПРОГРАММА:

```
/* Вычислить значение квадратного корня с заданной точностью */
#include <stdio.h>

main()
```

```
{
    float Number;          /* заданное число      */
    float Precision;       /* заданная точность */
    float Root;            /* корень              */

    /* Ввести и проверить исходные данные */
    while (1) {
            printf("Введите число для вычисления корня>");
            scanf("%g", &Number);
    if (Number > 0) break;
            printf("Недопустимое число./n");
            printf("Число должно быть больше нуля./n");
    }
    while (1) {
            printf("Введите точность вычисления корня>");
            scanf("%g", &Precision);
    if (Precision > 0) break;
            printf("Недопустимая точность./n");
            printf("Точность должна быть больше нуля./n");
    }
    /* Установить начальное значение корня */
    Root = 1;

    /* Вычислить значение квадратного корня */
    do {
            /* Вычислить следующее приближение корня */
            Root = (Number/Root + Root)/2;
    } while (abs(Number/(Root*Root)-1) > Precision);

    /* Вывести результат */
    printf("Квадратный корень из %g с точностью %g равен %g/n",
            Number, Precision, Root);
}
```

В этой программе мы использовали конструкцию цикла с постусловием. Оператор цикла с постусловием осуществляет проверку условия продолжения цикла после выполнения тела цикла, отсюда и название: оператор цикла с постусловием. Этот оператор выглядит следующим образом:

```
do {
    <оператор1>
        ...
    <операторN>
} while (<выражение>);
```

Сначала выполняются <оператор1>...<операторN>, затем вычисляется значение <выражения>. Если оно истинно, то снова выполняется тело цикла и вычисляется значение выражения. Как только значение выражения станет ложным, выполнение продолжится со следующего за оператором цикла оператора.

Так как значение выражения проверяется после выполнения тела цикла, то <оператор1>...<операторN> обязательно выполнятся хотя бы один раз.

При использовании этого оператора также необхсдимо помнить о том, что в теле цикла должны быть операторы, влияющие на значение выражения таким образом, чтобы после конечного числа повторений цикл завершился.

Результаты работы программы могут выглядеть, например, так:

Введите число для вычисления корня>25

Введите точность вычисления корня>0.1

```
Квадратный корень из 25 с точностью 0.1 равен 5
Введите число для вычисления корня>1
Введите точность вычисления корня>0.01
Квадратный корень из 1 с точностью 0.01 равен 1

Введите число для вычисления корня>2
Введите точность вычисления корня>0.00001
Квадратный корень из 2 с точностью 1e-05 равен 1.41421
```

ИСПОЛЬЗОВАНИЕ ПОДПРОГРАММ

В предыдущих задачах мы сначала разрабатывали алгоритм в виде небольших логически завершенных фрагментов (блоков), а затем собирали их вместе и получали полный алгоритм решения задачи. Но если задача достаточно сложная и алгоритм ее решения занимает не один лист, то работать с ним и с написанной по нему программой становится сложно. Поэтому хотелось бы иметь механизм, позволяющий оставить алгоритм в виде небольших блоков, и программу, также состоящую из небольших связанных между собой фрагментов, соответствующих блокам алгоритма. Такую возможность нам предоставляет механизм разбиения программы на так называемые подпрограммы. Они представляют собой отдельно описанные фрагменты и не приводятся в программе целиком, а в том месте, где они должны находиться, ставятся обращения к подпрограммам (или вызовы подпрограмм).

Используя этот механизм после этапа разработки алгоритма, мы решаем, какие фрагменты мы будем подставлять на их места, а какие оставим в виде отдельных подпрограмм. В те места, где они должны быть, вставим обращения к ним.

Подпрограммы обычно применяются в двух случаях. Во-первых, когда один и тот же фрагмент должен использоваться в разных частях программы. В этом случае нужно либо записывать несколько раз одно и то же, что очень нежелательно, либо вынести этот фрагмент в отдельную подпрограмму. Во-вторых, подпрограммы используют для того, чтобы бороться с большими размерами программы, потому что, как показывает практика, разобраться в нескольких маленьких программах легче, чем в одной большой.

В отличие от других языков программирования в Си используется только один тип подпрограмм - функция, т. е. подпрограмма, которой через ее параметры передаются некоторые значения и которая возвращает результаты своей работы через значение функции и специально выделенные для этой цели параметры.

Функция определяется заголовком и телом функции. Заголовок содержит имя функции, тип возвращаемого результата, имена и типы формальных параметров, заключенные в скобки и разделенные запятыми. Телом функции является составной оператор (блок), объединяющий описания внутренних переменных функции, и операторы, реализующие функцию.

Определение функции выглядит следующим образом:

```
<тип результата> <имя функции>(<список формальных параметров>)
{
   <тело функции>
}
```

Например:

```
double          /* тип результата                  */
linefunc        /* имя функции                     */
(               /* список формальных параметро в   */
    double x,   /* имя и тип первого параметра     */
    double a,   /* имя и тип второго параметра     */
    double b    /* имя и тип третьего параметра    */
)
```

```
{                          /* — тело функции —       */
    return (a*x+b);        /* возвращаемое значение */
}
```

Более привычной записью такой функции будет следующая:

```
double linefunc (double x, double a, double b)
{
    return (a*x+b);
}
```

В ранних реализациях языка Си тип параметров описывался по-другому. В списке формальных параметров указывались только имена переменных, а их тип описывался после параметров перед фигурной скобкой начала тела функции.

```
double linefunc(x, a, b)
double x;
double a;
double b;
{
    return (a*x+b);
}
```

Если тип результата функции (или, как обычно говорят, тип функции) не указан, то по умолчанию считается, что она имеет тип int.

Для реализации процедур в языке Си используются функции, не возвращающие значения. В поле типа результата таких функций рекомендуется писать ключевое слово void, указывающее, что функция не возвращает значения.

Например:

```
void errmsg(char *s)
{
        printf("*** ОШИБКА *** %s/n", s);
}
```

Эта функция печатает сообщение об ошибке, указанное аргументом. Вызов этой функции может иметь следующий вид:

```
errmsg("Деление на нуль");
```

Описание функции применяется в том случае, когда вызов ее в тексте программы встречается раньше, чем определение . Для того чтобы компилятор мог проверить правильность обращения к функции (если в описании указаны типы аргументов) и правильно сгенерировать коды для операций, в которых используется значение функции, необходимо перед использованием функции описать ее. Описание функции соответствует заголовку функции и содержит ее имя и тип результата , а также может содержать и типы аргументов.

Приведем примеры описания функции linefunc:

```
double linefunc();
double linefunc(double, double, double);
double linefunc(double x, double a, double b);
```

Используемый нами для включения в программы файл stdio.h как раз и содержит описания стандартных функций ввода/вывода.

Вызов функции - это выражение, значением которого является результат, возвращаемый функцией. Если вызов функции используется как отдельный оператор (т. е. не в некотором выражении), то значение, возвращаемое функцией, теряется. Если функция, описанная как не возвращающая значения (void), используется в некотором выражении, то возвращаемое ею значение не определено.

Существует два способа вызова функций:

<имя функции> (< список фактических параметров>)
или
(* < указатель на функцию>)(<список фактических параметров>)

Указателем на функцию является выражение, имеющее своим значением адрес некоторой функции.

Фактические параметры (аргументы) представляют собой список разделенных запятыми выражений, значения которых передаются функции.

В языках программирования обычно используют два механизма передачи параметров: по значению и по ссылке. При передаче параметра по значению аргументом может быть произвольное выражение, значение которого и передается в подпрограмму. При передаче параметра по ссылке аргументом может быть только переменная (как простая, так и структурированная). В этом случае в подпрограмму передается не значение переменной, а ее адрес, для того чтобы по нему могло бы быть занесено новое значение и тем самым изменено значение передаваемой в качестве параметра переменной.

В разных языках используются различные решения, в одних все аргументы передаются по ссылке (если аргумент является выражением, используются временные ячейки памяти, в которые заносятся значения выражений и в подпрограмму передаются их адреса); в других используются оба механизма, и тогда при указании списка параметров в определении функции используются специальные ключевые слова, показывающие, что значение аргумента может быть изменено в подпрограмме (т.е. необходимо осуществить передачу параметра по ссылке).

В языке Си использовано третье решение, т.е. все аргументы передаются по значению, а для передачи аргумента по ссылке в подпрограмму передается адрес требуемой переменной с помощью операции получения адреса объекта (&). Правда, при этом изменяется работа с данным аргументом в самой подпрограмме, потому что необходимо имеющийся параметр описать как указатель и использовать при доступе к нему операции работы с указателями, что несколько усложняет текст.

Указатели позволяют нам в одной переменной хранить адрес другой переменной. Для работы с указателями используются две операции: получение адреса переменной (&) и извлечение значения с использованием указателя (*). Для того чтобы при описании переменной сообщить компилятору, что она является указателем на значение некоторого типа, перед именем переменной ставят звездочку (*).

Примеры:

```
char ch;
char *cptr;
int val, *ivptr, n;
double r, *rp;
```

Здесь ch - переменная символьного типа; cptr - указатель на значение символьного типа; val и n - целые переменные; ivptr - указатель на значение целого типа; r - переменная вещественного типа двойной точности; rp - указатель на значение вещественного типа двойной точности.

Рассмотрим использование операций получения адреса (&) и извлечения значения по указателю (*). Так, оператор присваивания

```
ivptr = &val;
```

присваивает переменной ivptr адрес переменной val. После этого выражение *ivptr будет иметь то же значение, что и переменная val. Например, при присваи-

вании переменной val нового значения изменится и значение выражения *ivptr. И наоборот, при занесении некоторого значения по адресу, содержащемуся в переменной ivptr, например с помощью следующего оператора присваивания

```
*ivptr = 5;
```

это значение занесется в переменную val. Обратите внимание на то, что выражение *ivptr может появляться и слева от знака операции присваивания наравне с обычной переменной, потому данное выражение и соответствует переменной.

Оператор присваивания

```
n = *ivptr;
```

в нашем случае соответствует оператору

```
n = val.
```

Рассмотрим для примера функцию, меняющую местами значения своих аргументов (этот пример впервые был использован в широко известной книге Кернигана и Ритчи).

```
/* Поменять местами значения a и b */
void swap(int *a, int *b)
{
        int temp;            /* рабочая переменная                    */

        temp = *a;           /* запомнить в рабочей переменной значение,
                                на которое указывает a                */
        *a = *b;             /* занести в переменную, указываемую a,
                                значение, на которое указывает b      */
        *b = temp;           /* занести в переменную, указываемую b,
                                значение из рабочей переменной        */
}
```

Обращение к этой функции будет выглядеть следующим образом:

```
swap(&x, &y);
```

Порядок вычисления аргументов и порядок их загрузки в стек не гарантируются и зависят от реализации.

Во время выполнения функции не производится проверка числа и типа аргументов, переданных функции. Такая проверка может проводится при компиляции, если перед использованием функции располагается ее определение или описание, в котором указываются количество и типы аргументов.

Оператор возврата используется в языке для того, чтобы можно было закончить выполнение функции в любом месте программы, а не только при достижении конца тела функции и для возврата значения функции.

В языке Си используются два варианта оператора возврата: один для возврата управления из функций (не возвращающих значения), а другой для возврата значений. Записываются эти операторы следующим образом:

```
return;

return <выражение>;
```

Возвращаемое функцией значение должно соответствовать типу, указанному при определении и описании функции, проверка этого не производится.

В языке Си можно описывать переменные, которые содержат указатели на функции, т. е. адреса функций, и осуществлять обращение к функциям при помощи этих указателей.

Рассмотрим это на примере.

```
double y;
/* описание функции    */
double linefunc(double x, double a, double b);
```

```
double *funcptr();          /* описание переменной funcptr,
                               являющейся указателем на функцию,
                               возвращающую значение типа double */
funcptr = &linefunc;        /* присваивание переменной funcptr
                               адреса функции linefunc          */
y = (*funcptr)(2.4, 10., 7); /* вызов функции с
                               использованием указателя         */
```

Указатель на функцию можно передавать также в качестве аргумента в другую функцию.

Все функции в языке Си могут быть рекурсивными, т. е. любая из них может явно или косвенно вызывать саму себя.

Рассмотрим это на примере программы вычисления факториала.

```
/* Программа вычисления факториала */
/* N! = 1*2*3*...*(N-2)*(N-1)*N          */
#include <stdio.h>

unsigned long F(unsigned long n);

main( )
{
        unsigned long n, fac;

        printf("Введите N >");
        scanf("%u", &n);

        fac = F(n);

        printf("N! = %lu/n", fac);
}

unsigned long F(unsigned long n)
{
        return n == 1 ? 1 : n * F(n-1);
}
```

Пример.

```
/* Программа вычисления обьема сферы */
#include <stdio.h>

#define PI 3.14159

float sphere(int rad); /* описание функции */

main()
{
        float volume;      /* объем сферы   */
        float radius;      /* радиус сферы  */

        /* ввод радиуса */
        printf("Введите радиус >");
        scanf("%d", &radius);

        /* проверка введенного значения */
        if (radius < 0) {
```

```
            printf("ОШИБКА: Радиус должен быть положительным/n");
            exit(1); /* закончить программу с кодом возврата 1 */
        }

        /* вычисление обьема сферы */
        volume = shpere(radius);

        /* вывод результата */
        printf("Сфера радиуса %d имеет обьем %f/n");
}

/* вычисление обьема сферы по радиусу */
float sphere(int rad)
{
        float result; /* результат */

        /* вычисление по формуле V = (4*PI*R^3)/3 */
        result = rad * rad * rad;
        result = 4 * PI * result;
        result = result / 3;

        return result; /* возврат результата */
}
```

ПУЗЫРЬКОВАЯ СОРТИРОВКА

В этой главе мы рассмотрим программу упорядочения последовательности вещественных чисел методом пузырьковой сортировки.

ЗАДАЧА:
Упорядочить по возрастанию последовательность вещественных чисел.

ДАНО:
Количество элементов последовательности:
элемент1, элемент2, ...

ТРЕБУЕТСЯ:
Переставить элементы таким образом, чтобы второй элемент был не меньше первого, третий не меньше второго, четвертый не меньше третьего и т.д.

МЕТОД:
Для решения этой задачи будем использовать «метод пузырька», названный так за аналогию с пузырьком воздуха, всплыгающим в воде. Метод основывается на том, что попарно просматриваются все элементы последовательности. Если элемент с большим номером меньше элемента с меньшим номером, то они меняются местами (т.е. как бы более легкий (больший) пузырик всплывает вверх). Таким образом, после первого просмотра последовательности самый большой элемент окажется последним и далее его можно уже не рассматривать. Второй просмотр организуется аналогично первому и т. д. до тех пор, пока при очередном просмотре не будет сделано ни одной перестановки. Это значит, что все элементы последовательности уже упорядочены.

АНАЛИЗ:
1. Для решения задачи нам потребуется ввести последовательность элементов. Поступим так же, как и при нахождении минимального и максималь-

ного элементов: сначала введем количество элементов в последовательности, а затем будем вводить сами элементы.

2. В отличие от задачи нахождения минимального и максимального элементов, где каждый элемент мы использовали только один раз, в данном случае нам необходимо хранить в памяти все элементы последовательности, так как все они нужны для сравнений и вывода конечного результата.

Для хранения последовательностей элементов одного и того же типа используется специальная структура данных - массив. Он представляет собой блок последовательно пронумерованных объектов одного типа. Основные свойства массивов :

все элементы имеют один и тот же тип данных;
доступ к элементам осуществляется по номеру,
который часто называют индексом.

В языке Си описания массива и переменной отличаются тем, что вслед за идентификатором переменной в квадратных скобках указывается количество элементов в массиве, которое задается константным выражением. Если массив содержит больше одной размерности, то они указываются последовательно. Количество элементов по каждой размерности заключается в свою пару квадратных скобок ([<константное выражение>]).

Пример:

```
char str[20];        /* одномерный массив (вектор) из 20
                        элементов символьного типа              */
int val[3][50];      /* двумерный массив (матрица), содержащий 3
                        строки по 50 элементов в каждой, тип
                        элементов целый                         */
float res[8][5][4];  /* трехмерный массив вещественных чисел    */
```

Номера элементов массива всегда начинаются с нуля. Так в массиве из 20 элементов будут номера с 0 до 19.

Хранение многомерных массивов в памяти осуществляется по строкам, т.е. быстрее всего изменяется самый правый индекс.

Доступ к элементу массива осуществляется по индексу. Для этого после имени массива в квадратных скобках указывается выражение, значение которого задает индекс элемента. Для доступа к элементам многомерных массивов подряд указывается необходимое количество выражений, каждое в своих квадратных скобках.

Примеры:

```
str[3] = str[i+5];
val[i][j] = val[j][i];
res[1][k][0] = k1 * res[1][k][1] + k2 * res[1][k][2] +
    k3 * res[1][k][3];
```

Идентификатор массива без указания индексов соответствует адресу, с которого начинается размещение элементов массива в памяти. Если указаны не все индексы, требуемые для доступа к отдельному элементу, эта запись будет соответствовать адресу начала размещения в памяти соответствующего подмассива.

РАЗРАБОТКА АЛГОРИТМА:

Ввести исходные данные.
Упорядочить последовательность.
Вывести упорядоченную последовательность.

Так будет выглядеть начальный план нашей программы. В отличие от программы нахождения наименьшего и наибольшего элементов в данном случае

механизм решения задачи такой: сначала вводятся все исходные данные, затем они обрабатываются и после этого выводится результат. Очень многие программы, которые вам придется разрабатывать, будут подчиняться одному из рассмотренных механизмов обработки.

```
{Ввести исходные данные}
Ввести и проверить количество элементов последовательности.
Ввести элементы последовательности.

{Упорядочить последовательность}
Установить номер последнего просматриваемого элемента
последовательности, равным номеру последнего элемента
последовательности без единицы.
Повторять:
    Сбросить флаг перестановки.
    Выполнить просмотр подмассива до последнего
    просматриваемого элемента.
    Уменьшить на единицу номер
    последнего просматриваемого элемента.
пока установлен флаг перестановки.

{Выполнить просмотр подмассива до последнего просматрива-}
{ емого элемента }
Для индекса от нуля до последнего  просматриваемого элемента повторять:
    выполнить сравнение и перестановку соседних элементов.
конец повторения.

{Выполнить сравнение и перестановку соседних элементов}
Если элемент от индекса больше, чем следующий , то
    поменять элементы местами;
    установить флаг перестановки;
конец если.
```

АЛГОРИТМ:

```
    Ввести исходные данные.
    Упорядочить последовательность.
    Вывести упорядоченную последовательность.

    {Ввести исходные данные}
    Ввести и проверить количество элементов последовательности.
    Ввести элементы последовательности.

    {Упорядочить последовательность}
    Установить номер последнего просматриваемого элемента
    последовательности, равным номеру последнего элемента
    последовательности без единицы.
    Повторять:
        Сбросить флаг перестановки.
        {Выполнить просмотр подмассива до последнего просмат-}
        {риваемого элемента }
        Для индекса от нуля до последнего просматриваемого элемента повторять:
            {Выполнить сравнение и перестановку соседних элементов }
            Если элемент от индекса больше, чем следующий, то
                поменять элементы местами.
                установить флаг перестановки.
            конец если.
        конец повторения.
```

Уменьшить на единицу номер последнего просматриваемого элемента.
пока установлен флаг перестановки.

{Вывести упорядоченную последовательность}
Вывести элементы последовательности.

Итак, мы записали алгоритм не в виде одного блока, а разбили его на четыре подпрограммы.

ПРОГРАММА:

```
/* Программа упорядочивания последовательности вещественных */
/* чисел методом пузырьковой сортировки */
#include <stdio.h>

/* Максимальное количество элементов последовательности */
#define MAXQNT 100

void InpSeq(int *PQnt, float Array[]);
void SortSeq(int Qnt, float Array[]);
void OutSeq(int Qnt, float Array[]);

main()
{
        float Array[MAXQNT];
        int Qnt;

        /* Ввести исходные данные                      */
        InpSeq(&Qnt, Array);
        /* Упорядочить последовательность              */
        SortSeq(Qnt, Array);
        /* Вывести упорядоченную последовательность */
        OutSeq(Qnt, Array);
}

/* Ввод исходных данных */
void InpSeq(int *PQnt, float Array[])
{
        int i;

        /* Ввести и проверить количество элементов */
        /* последовательности                      */
        for(;;) {
            printf("Введите количество элементов>");
            scanf("%d",PQnt);
        if (*PQnt > 0 && *PQnt <= MAXQNT) break;
            printf("Недопустимое количество элементов./n");
            printf("Количество элементов должно быть больше нуля");
            printf(" и меньше %d/n", MAXQNT);
        }
        /* Ввести элементы последовательности */
        for(i = 0; i < *PQnt; i++) {
            printf("Введите %d-й элемент>", i);
            scanf("%g", &Array[i]);
        }
}

/* Упорядочивание последовательности */
```

```
void SortSeq(int Qnt, float Array[])
{
        int Last;           /* Номер последнего просматриваемого элемента*/
        int ChngFlag;       /* Флаг перестановки                          */
        int Index;          /* Индекс просматриваемого элемента           */
        float TempEl;       /* Временная ячейка для перестановки          */
                            /* элементов                                  */

        /* Установить номер последнего просматриваемого элемента */
        /* последовательности, равным номеру последнего элемента */
        /* последовательности без единицы                        */
        Last = Qnt-1;
        do {
            /* Сбросить флаг перестановки */
            ChngFlag = 0;
            /* Выполнить просмотр подмассива до последнего */
            /* просматриваемого элемента                   */
            for (Index = 0; Index < Last; Index++) {
                /* Выполнить сравнение и перестановку соседних */
                /* элементов                                   */
                if (Array[Index] > Array[Index+1]) {
                    /* Поменять элементы местами     */
                    TempEl = Array[Index];
                    Array[Index] = Array[Index+1];
                    Array[Index+1] = TempEl;
                    /* Установить флаг перестановки */
                    ChngFlag = 1;
                }
            }
            /* Уменьшить на единицу номер последнего */
            /* просматриваемого элемента             */
            Last—;
        } while (ChngFlag);
}

/* Вывод упорядоченной последовательности     */
void OutSeq(int Qnt, float Array[])
{
        int i;

        /* Вывести элементы последовательности */
        for(i = 0; i < Qnt; i++) {
            printf("%g/t", Array[i]);
        }
}
```

Результат работы программы может выглядеть, например, следующим образом:

```
Введите количество элементов>10
Введите 0-й элемент>1.0
Введите 1-й элемент>-3
Введите 2-й элемент>23.456
Введите 3-й элемент>-0.75
Введите 4-й элемент>12e3
Введите 5-й элемент>1E-3
Введите 6-й элемент>67
Введите 7-й элемент>45.8
Введите 8-й элемент>98.99
```

Введите 9-й элемент>1.2345
-3-0.750.00111.234523.45645.867 98.9912000

ОПЕРАЦИИ, ИХ ПРИОРИТЕТЫ И ПОРЯДОК ВЫПОЛНЕНИЯ

Все операции сведены в таблицу. Приоритет операций уменьшается в ней сверху вниз. Порядок выполнения операций одного приоритета указан в правом столбце.

Название операции	Знак	Порядок выполнения
Вызов функции	( )	слева направо
Выбор элемента массива	[ ]	слева направо
Выбор элемента структуры	.	слева направо
Выбор элемента структуры по указателю	->	слева направо
Изменение знака	-	справа налево
Побитовое дополнение	~	справа налево
Логическое отрицание	!	справа налево
Взятие значения по указателю	*	справа налево
Получение адреса объекта	&	справа налево
Инкримент	++	справа налево
Декримент	—	справа налево
Получение размера объекта	sizeof	справа налево
или типа		справа налево
Явное преобразование типа	(<имя типа>)	справа налево
Умножение	*	слева направо
Деление	/	слева направо
Взятие по модулю (вычисление остатка от деления)	%	слева направо
Сложение	+	слева направо
Вычитание	-	слева направо
Сдвиг влево	<<	слева направо
Сдвиг вправо	>>	слева направо
Меньше	<	слева направо
Больше	>	слева направо
Меньше или равно	<=	слева направо
Больше или равно	>=	слева направо
Равно	==	слева направо
Не равно	!=	слева направо
Побитовое «И»	&	слева направо
Побитовое «ИСКЛЮЧАЮЩЕЕ ИЛИ»		слева направо
Побитовое «ИЛИ»	|	слева направо
Логическое «И»	&&	слева направо
Логическое «ИЛИ»	| |	слева направо
Условное выражение	? :	справа налево
Присваивание	=	справа налево
Присваивание с умножением	*=	справа налево

Присваивание с делением	/=	справа налево
Присваивание с взятием по модулю	%=	справа налево
Присваивание со сложением	+=	справа налево
Присваивание с вычитанием	-=	справа налево
Присваивание со сдвигом влево	<<=	справа налево
Присваивание со сдвигом вправо	>>=	справа налево
Присваивание с побитовым «И»	&=	справа налево
Присваивание с побитовым «ИЛИ»	|=	справа налево
Присваивание с побитовым «ИСКЛЮЧАЮЩИМ ИЛИ»		справа налево
Последовательное вычисление		слева направо

Для явного указания порядка выполнения операций могут использоваться круглые скобки.

ЗАКЛЮЧЕНИЕ

Мы с вами прошли начальный этап изучения языка Си. Нами рассмотрены основные типы данных, управляющие конструкции и операции языка. Этого набора достаточно для написания широкого спектра программ. К сожалению, ограниченный объем выпуска не позволил нам рассмотреть более сложные элементы языка, но, может, это и к лучшему. Вы имеете возможность приобрести опыт программирования на изученном подмножестве языка. И тогда вам будет легче продолжить его изучение.

Не рассмотренные нами элементы языка мы вкратце приводим ниже.

НЕРАССМОТРЕННЫЕ ОПЕРАТОРЫ

Оператор выбора

Оператор выбора используется для выбора действия в зависимости от значения выражения, заданного в заголовке. Оператор выбора имеет следующий формат:

```
switch(<выражение>) {
case    <константа1>:
        <оператор1>
         ...
        <операторN>
case    <константа2>:
        <операторN+1>
         ...
        <операторM>
  .
  .
  .
default:
        <операторL>
         ...
        <операторK>
}
```

Метки и оператор перехода

Любой оператор может иметь метку для того, чтобы можно было перейти на него с помощью оператора перехода (goto). Метка состоит из идентификатора,

за которым стоит двоеточие (:). Область определения метки совпадает с телом той функции, в которой она определена.

Оператор перехода используется для передачи управления на помеченный оператор. Записывается оператор перехода следующим образом:

goto <метка>;

Этот оператор обычно используется для выхода из вложенных управляющих конструкций.

КЛАССЫ ПАМЯТИ И ОБЛАСТИ ДЕЙСТВИЯ ПЕРЕМЕННЫХ

По классу памяти переменные в языке Си подразделяются на статические (static) и динамические (auto, register). Статические переменные существуют весь период выполнения программы. Динамические переменные создаются при входе в функцию и уничтожаются при выходе из нее. Динамические переменные, в свою очередь, могут быть размещаемыми в памяти (auto) или регистровыми (register).

Класс памяти может указываться явно при описании переменной перед спецификацией типа. В том случае, когда класс памяти не указан явно, он определяется в зависимости от того места, где эта переменная описана. Переменные, описанные внутри некоторой функции, являются динамическими, размещаемыми в памяти. Переменные, описанные на одном уровне с определением функций, т.е. не находящиеся внутри никакой функции, являются статическими.

Переменные, описанные внутри некоторой функции, доступны для использования только внутри этой функции независимо от класса памяти.

Переменные, описанные на одном уровне с функциями, перед которыми не стоит ключевое слово static или стоит ключевое слово extern, доступны для использования в любом месте программы, даже если программа включает в себя несколько программных файлов.

Переменные, описанные на одном уровне с функциями, перед которыми стоит ключевое слово static, доступны для использования во всех функциях, но только того файла, в котором они описаны.

Слово static может использоваться в определении функции для указания того, что эта функция доступна только в пределах файла, в котором она определена.

НЕРАССМОТРЕННЫЕ ТИПЫ И СТРУКТУРЫ ДАННЫХ

Символьные и строковые данные

Для представления символьных и строковых данных в языке Си используется тип char. Объект этого типа может содержать один символ (буква, цифра, знак пунктуации или некоторый управляющий код).

Символьные константы состоят из одного символа, заключенного в апострофы (').

Примеры.

'A' 'a' '7' '$'

Для задания управляющих кодов используются так называемые специальные символы. Их список приведен в таблице.

Специальные символьные константы

Название	Обозначение	Символ
Новая строка	NL (LF)	'/n'
Горизонтальная табуляция	HT	'/t'
Вертикальная табуляция	VT	'/v'

Возврат на шаг	BS	'/b'
Возврат каретки	CR	'/r'
Звонок (сигнал)	BELL	'/a'
Апостроф	'	'/''
Кавычка	«	'/»'
Обратная наклонная черта	/	'//'
Пусто (нулевой символ)	NUL	'/0'

Кроме того, любой символ может быть представлен своим восьмеричным или шестнадцатиричным кодом. Для восьмеричного кода используется запись ' \ooo', где o - восьмеричная цифра, а для шестнадцатиричного кода - '\xhh', где h - шестнадцатиричная цифра. Например, ' \101' и '\x41' соответствуют букве 'A', а '\011', '\x09' или ' t' соответствуют символу горизонтальной табуляции.

Важно заметить, что символьные константы принадлежат типу int, но в них используется только один байт, поэтому они могут присваиваться переменным типа char.

Строковые константы записываются последовательностью символов, заключенных в кавычки.

Примеры.

```
"This is a character string"
"Это символьная строка"
"A" "12345" "0" "$"
```

В конце каждой символьной строки компилятор помещает нулевой символ ('/0'), отмечающий конец данной строки. Поэтому каждая строка занимает на один байт больше памяти, чем символов в строке. По этой же причине строка, содержащая один символ, не совпадает с символьной константой.

Для представления строк в языке Си нет специального типа данных. Они принадлежат типу массива символов длиной на единицу больше количества символов в строке.

Каждая строковая константа, даже если она идентична другой строковой константе, сохраняется в отдельном месте памяти.

Если две строковые константы записаны одна за другой, то во время компиляции они представляются одной строкой. Например:

"Hello," "Friend!\n"

эквивалентно одной строке

"Hello, Friend!\n"

Такой механизм представляет удобное средство для записи длинных строковых констант.

Структуры

Структура - это совокупность нескольких логически связанных переменных, возможно, различных типов, объединенных в одну группу, имеющую для удобства работы одно имя.

Основные свойства структуры:

переменные, входящие в структуру (их обычно называют полями) могут иметь различный тип;

доступ к элементу структуры осуществляется по имени.

В общем виде описание структуры выглядит следующим образом:

```
struct <имя структуры> {
    <описание поля1>
```

```
        ...
    <описание поляN>
} <список переменных>;
```

Описанием поля может быть как описание простой, так и структурированной переменной.

Имя структуры в описании может отсутствовать, но тогда для описания переменных, имеющих такую структуру, нельзя будет использовать сокращенное описание:

struct <имя структуры> <список переменных>;

В общем описании структуры может также отсутствовать список переменных, тогда отдельно описывается структура и отдельно определяются переменные с использованием сокращенной записи:

Примеры:

```
stuct date {                        /* дата   */
        unsigned char day;          /* день   */
        unsigned char month;        /* месяц */
        unsigned char year;         /* год    */
};

struct date curdate;                /* текущая дата   */
struct date birthfate;              /* дата рождения */
```

Доступ к полям записи, как мы уже отмечали, осуществляется по имени. Для этого после имени переменной ставят точку (.), за которой указывают имя требуемого поля.

Примеры:

```
f1.name          - фамилия, массив символов
f1.name[2]       - третья буква фамилии, символ
f1.sex           - пол, символ
f1.age           - возраст, целое беззнаковое
f1.salary        - зарплата, вещественное число
```

Битовые поля

Битовые поля представляют собой особый случай структур: выделение памяти под поля происходит не в соответствии с типом поля, а явно указывается количество бит, содержащееся в поле.

Описание битовых полей производится аналогично описанию структур:

```
struct <имя структуры> {
    <тип> <идентификатор> : <константное выражение>;
        ...
    <тип> <идентификатор> : <константное выражение>;
} <список переменных>;
```

Битовое поле обязательно должно быть целого беззнакового типа. Массивы битовых полей, указатели на битовые поля и функции, возвращающие битовые поля, не допускаются.

Константное выражение задает количество бит, выделяемых переменной. Биты для переменных выделяются последовательно, неименованное битовое поле, чей размер указан как нулевой, служит для выравнивания на границу машинного слова.

Пример:

```
stuct command {
        unsigned reg2 : 4;
        unsigned reg1 : 4;
        unsigned opcode : 8;
};
```

Доступ к битовым полям осуществляется точно так же, как к элементам структуры.

Объединения

Объединение - это переменная, которая может хранить (в разное время) значения разного типа и размера.

Основными свойствами объединения являются следующие:

объединение в конкретный момент времени хранит только одно значение;

доступ к значениям разного типа в объединении осуществляется по разным именам.

Описание объединения имеет следующий вид:

```
union    <имя объединения> {
        <описание поля1>
            ...
        <описание поляN>
}           <список переменных>;
```

К объединению можно отнести все, что мы говорили про описание и доступе к стуктурам.

Рассмотрим пример описания и доступа к объединению.

```
union u_tag {
        int ival;
        float fval;
        char *pval;
} uval;
```

uval.ival - значение целого типа;
uval.fval - значение вещественного типа;
uval.pval - указатель на символ.

Перечисления

Полное описание перечисления выглядит следующим образом:

```
enum      <имя перечисления>
        {<список идентификаторов>} <список переменных>;
```

Варианты описания перечислений могут быть такие же, как и при описании структур.

<Список идентификаторов> представляет собой разделенный запятыми список идентификаторов, возможно, с заданными значениями. Значения идентификаторам присваиваются последовательно, начиная с нуля. Если какому-то идентификатору присваивается значение явно, то следующий за ним получит значение на единицу больше и т. д.

Переменные из <списка переменных> могут принимать только указанные в списке идентификаторов значения.

Описание перечислений покажем на примере:

```
enum day {saturday, sunday = 0, monday, tuesday, wednesday,
            thursday, friday
} workday;
```

Приведенные идентификаторы получат следующие значения: saturday - 0, так как это первый идентификатор; sunday - 0, потому что ему явно указано

значение; monday - 1, на единицу больше предыдущего и т. д., tuesday - 2, wednesday - 3, thursday - 4, friday - 5.

При указании явного значения идентификатора можно использовать произвольное константное выражение.

Ключевые слова языка Си

Ключевые слова - это идентификаторы, зарезервированные в языке Си, которые можно использовать в том смысле, как они определены в языке.

Список ключевых слов:

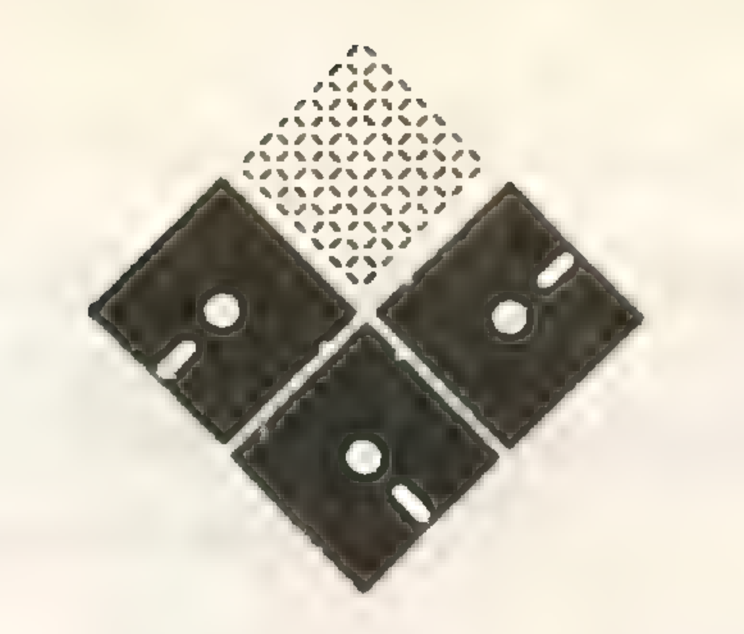
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Кроме указанных, в некоторых реализациях ключевыми могут быть еще слова:

asm cdecl far fortran huge near pascal

Литература

1. *Керниган Б., Ритчи Д., Фьюэр А. Язык программирования Си. Задачи по языку Си./Пер. с англ. - М.: Финансы и статистика, 1985. - 279 с., ил.*
2. *Уэйт М., Прайта С., Мартин Д. Язык Си. Руководство для начинающих./ Пер. с англ. - М.: Мир, 1988. - 512 с., ил.*
3. *Болски М.И. Язык программирования Си. Справочник./ Пер. с англ. - М.: Радио и связь, 1988. - 96 с., ил.*
4. *Хэнкак Я., Кригер К. Программирование на языке Си./ Пер. с англ. - М,: Радио и связь, 1980.*
5. *Языки программирования Ада, Си, Паскаль. Сравнение и оценка. - М.: Радио и связь, 1989.*



Система Турбо-Си объединяет в себе мощный компилятор языка Си и интегрированную среду для разработки и отладки программ. Необходимость такого рассказа видится в следующем. Скорее всего, вы заняты изучением языка программирования Си или планируете в ближайшем будущем взяться за это ответственное дело. При изучении языка, тренировке в написании и отладке на ЭВМ собственных программ возникнут по меньшей мере два вопроса: как пользоваться имеющимся компилятором и насколько «понимаемая» данным компилятором версия языка Си отличается от других версий. Эти важные вопросы рассмотрены в статье.

А.А.Ковалев

ТУРБО СИ ДЛЯ НАЧИНАЮЩИХ

ANSI СТАНДАРТ ТУРБО СИ

Вы уже знаете, что язык программирования Си разработал Денис Ритчи. Он реализовал Си для компьютера типа PDP фирмы DEC, который работал под управлением операционной системы UNIX. Эта версия приобрела большую известность благодаря классическому труду Кернигана и Ритчи «Язык программирования Си» (впервые выпущенному в 1978 году и переведенному на русский язык в 1983) и стала на первое время своеобразным стандартом языка Си. Данную версию принято называть K&R - по первым буквам фамилий классиков.

Однако язык Си продолжал развиваться. Особенно много новых версий компиляторов появилось для микрокомпьютеров. Большинство этих версий сохраняло совместимость с K&R по меньшей мере снизу вверх (т.е. программы, написанные с соблюдением классического стандарта, выполнялись на новых компиляторах). Но уже скоро встала проблема стандартизации всех тех улучшений и доработок, которые реализовывались в Си-компиляторах.

С этой целью летом 1983 года в институте ANSI был создан специальный комитет, получивший название X3J11 (или ANSI C Standart Committee). 29 сентября 1988 года комитет утвердил соответствующий стандарт, который стал надмножеством стандарта K&R и призван, по словам заместителя председателя комитета Томаса Палма, «дать программистам уверенность в том, что их Си-программы будут работать в любом новом окружении». Комитет не просто подготовил стандарт - было предложено 79 тестов, позволяющих быстро и эффективно определить степень совместимости со стандартом для каждого Си-компилятора.

Калифорнийская фирма «Борланд» (Borland International, CA), начав разрабатывать свой компилятор для персональной ЭВМ, поставила перед собой несколько задач. Во-первых, реализовать стандарт ANSI так, чтобы программисты могли иметь в своем распоряжении самый современный компилятор языка Си. Во-вторых, создать по возможности быстрый и эффективный компилятор, удовлетворяющий запросам профессиональных программистов. Наконец, создать дружественную пользователю среду, которая позволила бы максимально автоматизировать процесс разработки программ. В результате общественности была представлена система Турбо Си, которая, по мнению большинства экспертов, успешно разрешала поставленные задачи.

Си-компилятор фирмы «Борланд» работает быстро и эффективно. Из предлагаемых на рынке продуктов он наиболее соответствует стандарту ANSI (из уже упомянутых 79 тестов выполняются 60). Более того, предложенная интегрированная среда (получившая название Турбо-среды) для разработки программного обеспечения оценена как новый шаг в развитии компьютерного дела. Наконец, фирма не останавливается на достигнутом - постоянно появляются новые версии Турбо-Си, реализующие более совершенные методы подготовки и отладки программ, а

также новые библиотеки процедур и функций. Все это делает Турбо-Си одним из наиболее популярных языков программирования и среди «любителей», и среди профессионалов.

Турбо-Си - достаточно сложный язык, и для его полного понимания и профессионального использования необходим весь объем информации, который содержится в многочисленных фирменных руководствах и специально посвященных Турбо-Си книгах. Вполне понятно, что мы дадим только некую «выжимку» из этих источников, уделив внимание разработке программного обеспечения с использованием Турбо-среды и изменениям, имеющимся в Турбо-Си по сравнению со стандартом K&R. Остальное - в ваших руках.

Появляющиеся новые версии во многом требуют новой информации и даже новых подходов к программированию. В статье рассматривается версия 2.0 Турбо-Си. В последнее время появился пакет следующего поколения: Турбо Си++, отличающийся от предшественников новой интегрированной средой и новыми расширениями языка в сторону технологии объектно-ориентированного программирования. А впереди - все более мощные и современные продукты.

ТУРБО СИ - СОВРЕМЕННЫЙ СТИЛЬ ПРОГРАММИРОВАНИЯ

Еще раз подчеркнем, что Турбо Си был разработан «во исполнение» стандарта ANSI C Standart Commitee. Вполне правомерно ожидать, что Турбо Си вносит некоторые дополнения к классическому K&R, языку Си и соответствующему стилю программирования. Этот раздел ознакомит вас с некоторыми нововведениями, которые помогают в написании более удобных и «профессиональных» программ.

Мы не можем поставить перед собою цель подробно изложить новый стандарт Си, будучи ограниченными объемами настоящей публикации. Попробуем, однако, дать понятие о наиболее ярких и существенных доработках, которые упоминаются в руководстве пользователя Си фирмы «Борланд» (заинтересовавшись вопросом, вы безусловно сможете найти более подробное описание «Турбо-стандарта» языка Си).

Прежде всего заметим следующее. Турбо Си позволяет следовать наравне классическому (K&R) и современному (сам Турбо Си) стилям. Программы, написанные на K&R-Си, должны успешно выполняться на вашем компьютере.

Ключевые слова Турбо Си

Турбо Си использует некоторые ключевые слова, которые не предусмотрены в стандарте K&R. Вы должны учесть, что использование этих ключевых слов в программе в качестве ваших идентификаторов ни к чему хорошему не приведет. Поэтому сразу же приведем их полный список.

Однако надо оговориться. Турбо Си вводит не только ключевые слова, предусмотренные ANSI, но и небольшой ряд своих собственных, оригинальных. Но при этом вы можете установить в параметрах интегрированной среды (см. далее порядок работы с меню Options) запрет на использование несовместимых с ANSI ключевых слов и целиком следовать стандарту.

Список ключевых слов, не встречающихся в K&R, но предусмотренных стандартом ANSI, следующий:

```
const      void
enum       volatile
signed
```

Далее следует список оригинальных ключевых слов Турбо Си (употребление которых можно отключить):

```
asm         _cs     _DH
cdecl       _ds     _DL
_es         _DX
_ss         _BP
far         _AH     _DI
huge        _AL     _SI
interrupt   _AX     _SP
near        _BH
pascal      _BL     _CL
_CH         _CX
```

Важное ПРИМЕЧАНИЕ: ключевые слова entry и fortran, упомянутые в

K&R, не используются в Турбо-Си. А теперь перейдем к некоторым нововведениям.

Использование прототипов функций

В классическом стиле программирования на Си вы можете использовать объявление функции, состоящее только из имени, типа функции и пустых скобок. Параметры функции, если они есть, указываются только при непосредственном определении самой функции. Например, вы можете объявить в программе некую функцию swap следующим образом:

```
int swap();
```

Следующее за этим определение самой функции выглядит так:

```
int swap(a,b)
int *a, *b;
{
/* Тело функции */
}
```

Результат такого определения - отсутствие контроля над ошибками при использовании параметров. Руководство по программированию на Турбо Си советует избегать такого стиля.

ANSI стандарт и, естественно, Турбо Си допускают использование так называемых прототипов функций для объявления функций в вашей программе. Предлагается специальная форма описания функций, включающая информацию о параметрах функции. Компилятор использует эту информацию для проверки вызовов функций и для преобразования типов. Переопределим swap, используя прототипы (современный стиль программирования):

```
int swap(int *a, int *b);
```

Теперь при выполнении программа получает всю необходимую информацию, требуемую для выполнения контроля над ошибками при любом обращении к swap. Теперь вы можете использовать простой формат и при определении функции:

```
int swap(int *a, int *b)
{
    /* тело функции */
}
```

Современный стиль программирования повышает вероятность обнаружения ошибки, даже если вы не используете прототипы функций; если же прототипы используются, то компилятор автоматически отслеживает и обеспечивает соответствие описаний и определений.

Допускаются различные способы объявления функций. Если функция не имеет аргументов, ее можно объявить следующим образом:

```
int f(void);
```

- ключевое слово void мы рассмотрим в следующем разделе.

Если аргументы есть, обычно их указывают в скобках, разделяя запятыми. Но объявление можно сделать и без идентификаторов, например:

```
int func1(int *, long);
```

что соответствует более привычному

```
int func1(int *count, long total);
```

Прототип определяет Си-функцию как имеющую фиксированное число параметров. Для функций, имеющих различное значение, прототип может заканчиваться многоточием (...), например:

```
f(int *count, long total, ... );
```

В таком случае фиксированные параметры проверяются во время компиляции, а переменные передаются как при отсутствии прототипа.

Описание функции void

Еще одним усовершенствованием описаний функций является вводимый в Турбо Си и предусмотренный стандартом ANSI тип void. Он используется для явного описания функций, не возвращающих значений.

В начальной версии языка Си каждая функция возвращала значение некоторого типа; если же тип не был описан, то по умолчанию функ-

ции присваивался тип int. Подобно этому функция, возвращающая «сгенерированные» (нетипичные) указатели, обычно описывалась как возвращающая указатель типа char, только потому, что она должна была хоть что-то возвращать. В Турбо Си такое противоречие преодолено.

Аналогично используется слово void при описании прототипов функций (см. выше). Если функция имеет пустой список параметров, вместо него ставится слово void. Пример функции, которая «ничего не принимает и ничего не возвращает»:

```
void putmsg(void)
{
            printf("Hello, word/n");
}

main()
{
   putmsg();
}
```

Более того, можно преобразовывать выражения к типу void, чтобы явно указать - возвращаемое значение функции вам не нужно и поэтому игнорируется. Пример из руководства пользователя: для приостановки выполнения программы до нажатия какой-либо клавиши можно использовать выражение

```
(void) getch();
```

Можно объявить указатель типа void, данный указатель не будет указателем на ничто - просто создается указатель на некий объект, тип которого нет необходимости определять. Этому указателю можно присваивать значение любого другого и обратно, однако вы не можете использовать оператор косвенной адресации (*), так как используемый тип неопределен.

Модификаторы типа функции

Турбо Си определяет три модификатора типа, которые могут применяться только к функциям. Это модификаторы **pascal**, **cdecl** и **interrupt**. Стандарт ANSI не определяет эти модификаторы, однако Турбо Си все же обеспечивает их для того, чтобы наилучшим образом использовать возможные преимущества среды программирования ПЭВМ. Мы только кратко остановимся на назначении данных модификаторов - для обстоятельного их изучения требуется более серьезный разговор.

Модификатор типа pascal указывает компилятору использовать так называемые Pascal-подобные соглашения о передаче параметров аргументов функции, а не обычный Турбо Си способ. Этот модификатор допускает две возможности. Во-первых, можно писать функции в Турбо Си, которые будут использовать другие компиляторы. Во-вторых, можно использовать библиотечные подпрограммы Pascal-компилятора. Например, приведенная ниже версия функции int_pwr() может быть скомпилирована для использования Pascal-компилятором:

```
/* компилировать для Pascal-компиляторов */
pascal int_pwr(m,e)
int m;
register int e;
{
     register int temp;
     temp=1;
     for(;e;e—) temp*=m;
     return temp;
}
```

Для компилирования всех функций в файле так, чтобы они были типа pascal, не используя модификатор pascal, вы могли бы установить соответствующий параметр в меню Option.

Модификатор типа cdecl - противоположность модификатора pascal. Он предписывает Турбо Си компилировать функцию так, чтобы ее параметры передавались способом, который совместим с другими Си-функциями. Использовать cdecl следует только в тех случаях, когда в опциях интегрированной среды Турбо Си установлен режим компиляции с использованием Pascal-соглашения о вызовах, а в программе есть ряд функций, которые вы не хотите компилировать в pascal-формате.

Модификатор типа interrupt сообщает Турбо Си, что функция, которую он модифицирует, будет использоваться в качестве обработчика

прерываний. Разработку и установку обработчиков прерываний в этой статье не будем обсуждать, здесь необходимо использование дополнительных источников информации как по Турбо Си, так и по MS-DOS.

Тип enum

Турбо Си поддерживает перечисленный тип ANSI стандарта. Перечисленный тип используется для описания дискретной последовательности целых значений, например:

```
enum days  {sun, mon,
tues, wed, thur, fri, sat};
```

В days фактически заносятся целые константы, первая устанавливается в 0 (sun=0), последующие соответственно увеличиваются на 1 (mon=1 и т.д.). Но можно явно присвоить константам определенные значения; имена, не имеющие своих определенных значений, как и прежде, будут иметь предыдущие значения, увеличенные на 1.

```
enum coins {penny=1,
nickle=5,  dime=10, quater=25};
```

Можно использовать оператор typedef для определения нового типа - day (к примеру).

```
typedef  enum (sun, mon,
tues, wed, thur,fri,sat)  days;
```

После этого можно описать переменные, принадлежащие к абстрактному типу данных days.

Модификаторы const и volatile

В Турбо Си предусмотрены два модификатора типа, которые используются для управления способами доступа к переменным. Эти модификаторы называются const и volatile.

Модификатор const применяется для объявления переменной, значение которой ваша программа не может изменять (за исключением того, что вы можете задать первоначальное значение этой переменной). Например:

```
const float version = 3.20;
```

создает float (с плавающей точкой) переменную, названную version, которую модифицировать нельзя. Однако можно использовать переменную version в других типах выражений.

Таким образом, объявленная с модификатором const переменная будет получать свое значение из явной инициализации. Применение модификатора const к объявлению переменной гарантирует, что другие части вашей программы не будут модифицировать эту переменную.

Переменные типа const имеют одно очень важное свойство: они могут защищать аргументы функции от изменения их этой же функцией. В тех случаях, когда программа передает указатель на функцию, возможно, что эта функция модифицирует фактическую переменную, на которую указывает этот указатель. Однако если специфицировать указатель в объявлении параметра как const, код функции будет неспособен модифицировать то, на что он указывает. Например, функция code() в приведенной ниже короткой программе сдвигает каждую букву в сообщении на одну, т.е. А становится В и т.д. Использование const в объявлении параметра гарантирует, что код внутри этой функции не сможет модифицировать объект, на который указывает этот параметр.

```
void code( );
main( )
        {
code("это тест");
        }

        void code(str)
        const char *str;
        {
while(*str) printf("%c", (*str++)+1);
        }
```

Если из каких-то соображений вы написали функцию code() таким образом, что аргумент может быть в ней модифицирован, Турбо Си не будет ее компилировать и выдаст информацию об ошибке.

(Продолжение в одном из последующих выпусков)

ЖУРНАЛ "БЮЛЛЕТЕНЬ-БК"

С января 1990 года "Импульс" издает ежемесячный информационно-справочный журнал для пользователей компьютера БК. Он зарегистрирован как официальное всесоюзное издание.

В каждом номере журнала публикуется подробное описание до 100 новых программ, информация о малоизвестных программах, инструкции, листинги программ, описания доработок БК, статьи о "секретах" программирования, ответы на вопросы читателей.

Начата публикация учебника по программированию в кодах на БК для начинающих, который в отличие от всех других построен исключительно на практических материалах. Печатается и полное техническое описание БК, которое ранее нигде не публиковалось.

Для пользователей УКНЦ и БК-11 помещается бесплатное приложение.

Стоимость номера с пересылкой 6 руб.40 коп., годовая подписка - 64 руб. (скидка 15%). С 1992 года журнал занесен в каталог "Союзпечати", в 1991 году его можно заказать наложенным платежом. Условия подписки заказывайте по адресу: 252142 Киев-142, а/я 14, редакция журнала "Бюллетень-БК". Тел. 444-45-50.

Наша марка - "Импакт"

Научно-техническое малое предприятие "Импакт" - преемник объединения "Импульс", которое одним из первых в стране взяло на себя обслуживание пользователей БК-0010(-01) и получившего широкую известность благодаря наиболее полному и систематизированному фонду программного обеспечения для этих компьютеров.

"Импакт" поставляет индивидуальным пользователям и организациям.

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ КОМПЬЮТЕРОВ БК-0010(-01)

Программы поставляются в комплектах, среди которых особо популярные "25 и 50 лучших игр в кодах", "Основной комплект", "Минимакс" (наиболее качественные программы для БК). Раз в два месяца выпускаются комплексы "Лучшие новинки". Поставляются также отдельные программы.

Стоимость программы в комплекте 4 руб. В отличие от других организаций, "Импакт" не только поставляет программы, но и предоставляет БЕСПЛАТНЫЕ КОНСУЛЬТАЦИИ заказчикам.

Организацией разработан "Каталог-справочник программ для БК-0010(-01)". Каталог включает краткое описание содержания и данные о 1300 программах. Цена 17 руб. с пересылкой, для предприятий и организаций 49 руб.

"ИМПАКТ" - ПЕРВАЯ И ПОКА ЕДИНСТВЕННАЯ ОРГАНИЗАЦИЯ В СТРАНЕ, ПОСТАВЛЯЮЩАЯ ПРОГРАММЫ ДЛЯ КОМПЬЮТЕРА БК-11

С авторами программ заключаются договоры на тиражирование.
Детальный проспект об услугах высылается бесплатно.

Не забудьте пометить в своей записной книжке наш адрес:
252124 Киев-124, а/я 8, "Импакт".
Тел. 444-45-50, 440-91-20. 432-17-61, факс 227-08-37 (код 0-44).

"ИМПАКТ": ТРИ ГОДА РАБОТЫ В УСЛОВИЯХ РЫНКА - ГАРАНТИЯ УСПЕХА

Я53 **Язык программирования Си.** — М.: Знание, 1991. — 48 с. — (Новое в жизни, науке, технике. Сер. "Вычислительная техника и ее применение"; № 4).

ISBN 5-07-001720-9

35 к.

В этом выпуске доступно, с привлечением многочисленных примеров рассказано об одном из популярных языков программирования — языке Си. Этот язык используют системные программисты и разработчики прикладных программ.

Рассчитан на широкий круг читателей.

2302030000 **ББК 32.85**



Научно-популярное издание

ЯЗЫК ПРОГРАММИРОВАНИЯ СИ

Гл.отраслевой редактор *Г. Г. Карвовский*
Редактор *Б. М. Васильев*
Мл.редактор *Н. А. Васильева*
Художник *В. Н. Конюхов*
Худож.редактор *И. А. Емельянова*
Техн.редактор *Т. В. Луговская*
Корректор *В. И. Гуляева*

ИБ № 11303

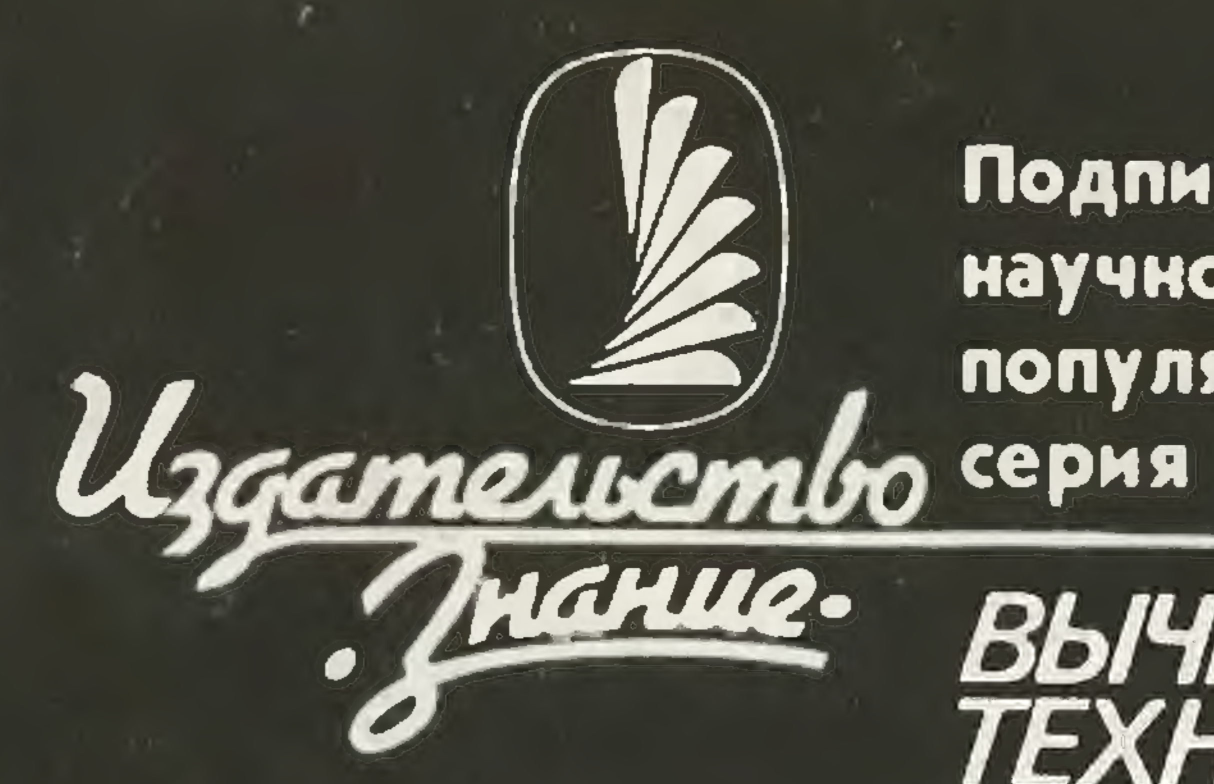
Подписано к печати 26.3.91. Формат бумаги 70x100 1/16. Бумага офсетная. Печать офсетная. Усл.печ.л.3,90. Усл.кр.-отт.8,45. Уч.-изд.л.3,21. Тираж 49965 экз. Заказ 2017. Цена 35 коп. Издательство "Знание". 101835, ГСП, Москва, Центр, проезд Серова,д.4. Индекс заказа 914704. Отпечатано с оригинал-макета издательства "Знание" на ордена Трудового Красного Знамени Тверском полиграфическом комбинате Государственного комитета СССР по печати. 170024, г.Тверь, пр.Ленина,5.

Цена 35 коп.

Индекс 70195

Адрес подписчика:

Солнечная 5-27



Подписная научно-популярная серия

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ЕЁ ПРИМЕНЕНИЕ

Я не испытываю страха перед ЭВМ. Я страшусь их отсутствия.

Айзек Азимов

Привилегия быть программистом в XX веке эквивалентна привилегии быть грамотным в XVII веке.

А.П.Ершов

Наш адрес: 101835, Москва, Центр, проезд Серова, 4